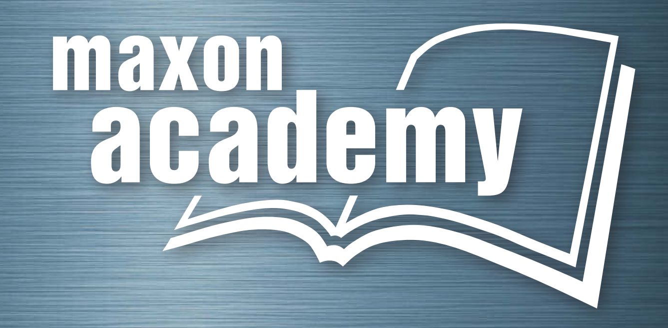


Urs Kafader

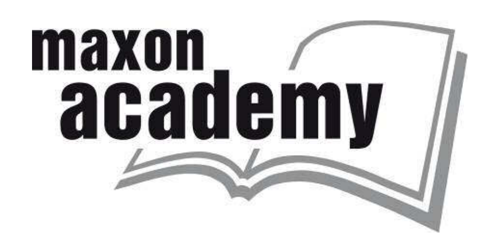
Motion Control für Einsteiger.

Mit maxon EPOS2 P.



maxon motor

driven by precision



Erste Ausgabe 2014

Version 1.2, Februar 2014

Motion Control für Einsteiger

mit maxon EPOS2 P

Herangehensweise

Dieses Lehrbuch verfolgt grundsätzlich eine praktische und eine experimentelle Herangehensweise; allerdings, in umgekehrter Reihenfolge wie die meisten anderen. Statt zuerst die Motion Control Theorie zu erklären und diese dann an Beispielen anzuwenden, beginnen wir hier mit praktischen Experimentieren an einem maxon *EPOS2 P* Positioniersystem mit Hilfe der *EPOS Studio* Software. All die relevanten Motion Control Prinzipien und Eigenschaften werden erklärt, wenn sie auftreten. Der Text besteht darum zum grossen Teil aus Übungen und praktischen Anleitungen; Hintergrundinformation findet man in den farbigen Kästchen.

Dies ist primär ein Lehrbuch über Motion Control und nicht über die Programmierung einer SPS. Deshalb ist der Teil über die Programmierung auch als Einführung zur Programmierung von Motion Control Aspekten und nicht als vollständiger SPS Programmier-Lehrgang zu verstehen.

Motion Control ist Mechatronik, die Kombination von Mechanik und Elektronik, von Aktuatoren (Motoren) und Sensoren, und alles geregelt und überwacht durch ein Softwareprogramm. In diesem Lehrbuch identifizieren und definieren wir die Rolle, das Verhalten und die Interaktion der verschiedenen Elemente eines Motion Control Systems. Dazu brauchen wir den internen Aufbau und die grundlegenden Arbeitsprinzipien nur beschränkt zu kennen. Nach Möglichkeit verwenden wir die Black-Box-Betrachtung. Welche Ausgaben werden für vorgegebene Eingangsgrössen erzeugt und welche Parameter beeinflussen die Ausgabe? So erklären wir zum Beispiel nicht, wie eine Vorsteuerung implementiert sein muss, sondern wir legen die dahinter liegende Idee dar und wie sich die Parameter während des Tuning verändern.

Sprache

Die Dokumentation und Software für die maxon EPOS Systeme stehen nur auf Englisch zur Verfügung. Wir versuchen im Lehrbuch die deutschen Begriffe zu verwenden. Die englische Übersetzung wird in Klammern hinzugefügt, soweit dies für das Verständnis der Abbildungen und die Bedienung nötig sind.
Eingedeutschte und häufig verwendete Begriffe wie *Computer*, *Encoder, Motion Controller, Feedback, …* werden wir nicht übersetzen. Ebenfalls wird die Einheit für die Drehzahl als rpm (*rounds per minute*) belassen, wie sie in der gesamten Dokumentation und im *EPOS Studio* erscheint; und nicht in UpM (Umdrehungen pro Minute) oder min^{-1} übersetzt.
Das *EPOS Studio* greift für die Beschriftung der Standard-Schaltflächen auf die eingestellte Sprache des Computers zurück. Somit kann es sein, dass ein *Next* auf deinem PC als *Weiter* bezeichnet wird.

Inhaltsverzeichnis

Motion Control für Einsteiger 3
Herangehensweise 3
Sprache 3
Kästchen 6
Teil 1: Vorbereitung 7
1 Das EPOS2 P Starter Kit vorbereiten 7
1.1 Installation der EPOS Studio Software 8
1.2 Kabelanschlüsse 10
1.3 Was befindet sich im schwarzen Gehäuse? 11
1.4 Das EPOS Studio starten 14
2 Das Motion Control System vorbereiten 19
2.1 Startup Wizard 19
2.2 Tuning 28
Teil 2: Der Motion Controller 35
3 Den Motion Controller erkunden 36
3.1 Profile Position Mode 36
3.2 Referenzfahrt (*Homing*) 44
3.3 Position Mode 46
3.4 Profile Velocity Mode 48
3.5 Velocity Mode 51
3.6 Current Mode 52
4 Das Werkzeug I/O Monitor 53
4.1 Ausgänge 53
4.2 Eingänge 54
5 Alternative Betriebsmodi (optional) 57
5.1 Analoger Sollwert 57
5.2 Master Encoder Mode: Elektronisches Getriebe 59
5.3 Step Direction Mode 59
Intermezzo: CAN und CANopen 61
6 Eine Einführung in CANopen und CAN 61
6.1 CAN 62
6.2 CANopen 62
6.3 CANopen Geräteprofil 63
6.4 Das Object Dictionary Tool der EPOS2 [internal] 67
6.5 Das Objektverzeichnis der EPOS2 P 70
6.6 Parameter Up- und Download 70
6.7 Kommunikation 71

Teil 3: Die SPS (Speicherprogrammierbare Steuerung)... 73
7 Das Programmierwerkzeug starten 74
7.1 Sample Project: Simple Motion Sequence 75
7.2 Die Projektdateien 79
8 Mein erstes Programm: *AxisMotion* 88
8.1 Vorbereitungsarbeiten 90
8.2 Die Hauptschritte programmieren 94
8.3 Weitere Bewegungen programmieren 106
9 Homing und IOs 110
9.1 Die Ein- und Ausgänge konfigurieren 110
9.2 Referenzfahrt mit Endschalter 112
9.3 Ausgänge gemäss dem Achszustand setzen 113
9.4 Eingänge lesen 115
10 Selbst-definierte Funktionsbausteine 116
10.1 Einen Funktionsbaustein erzeugen 116
10.2 In Function Block Diagram (FBD) programmieren 117
10.3 Selbstdefinierte Funktionsbausteine benützen 120
11 Anhang 121
11.1 Motor- und Encoder-Datenblätter 121
12 Verweise, Glossar 124
12.1 Liste der Abbildungen 124
12.2 Liste der Kästchen 126
12.3 Literatur 128
12.4 Index 129

Kästchen

Die farbigen Kästchen enthalten zusätzliche allgemeine Hintergrundinformationen über Motion Control und Programmierung, sowie spezielle Aspekte über das verwendete System und die Software.



Hintergrund zu Motion Control
Allgemeine Informationen und Grundlagenwissen zu Motion Control Konzepten.



IEC
Allgemeine Informationen und Hintergrundwissen über SPS-Programmierung mit IEC 61131-3.



EPOS Info
Zusätzliche Angaben zu speziellen Eigenschaften und über das Verhalten von *EPOS2 P* Systemen.



OpenPCS Info
Zusätzliche Angaben zu speziellen Eigenschaften und über das Verhalten der *OpenPCS* Software.



Best Practice
Tipps und Hinweise, wie man das *EPOS2 P* System, die *EPOS Studio* und *OpenPCS* Software effizient benützt und bedient.

Teil 1: Vorbereitung

Diese ersten beiden Kapitel dienen dazu, das System kennenzulernen und den Arbeitsplatz vorzubereiten.
In Kapitel 1 geben wir eine Übersicht über das physische System, wir installieren die *EPOS Studio* Software und erklären kurz die wesentlichen Elemente des Motion Control Systems.

In Kapitel 2 bereiten wir das Motion Control System für die darauf folgenden Übungen vor. Die Eigenschaften von Motor und Encoder müssen eingestellt und der Regelkreis getunt werden.

1 Das EPOS2 P Starter Kit vorbereiten



Abbildung 1: Der Inhalt des EPOS2 P *Starter Kit.*

Im *EPOS2 P* Starter Kit Koffer befindet sich eine Grundplatte mit einer “Black Box”, einem Motor mit Encoder, und einer Leiterplatte mit Schaltern, LEDs, und Potentiometern. Ebenfalls sind ein DC-Netzgerät und die nötigen Kabel enthalten.

1.1 Installation der EPOS Studio Software

Zuerst gilt es den Computer vorzubereiten. Im Starter Kit Koffer findet sich die *EPOS Positioning Controller* DVD mit all der benötigten Software. Installiere mit Hilfe der DVD die Software auf deinem Computer. Folge einfach den Anweisungen des Installationswizards.

Die Installation umfasst alle nötigen Informationen und Werkzeuge (wie zum Beispiel Handbücher, Firmware, Windows DLLs, Treiber), die für die Installation und den Betrieb des *EPOS2 P Programmable Positioning Controller* nötig sind. Das wichtigste Werkzeug ist das *EPOS Studio.* Es ist die Bedienoberfläche für den vollen Zugriff auf alle Eigenschaften und Parameter des Motion Controller.

Installation

Schritt 1 Setze die *EPOS Positioning Controller* DVD ins Computer-Laufwerk ein.
Autorun beginnt automatisch. Falls nicht, suche die Installationsdatei mit dem Namen *EPOS Positioning Controller.msi* mit dem Explorer. Doppelklick startet die Installation.

Schritt 2 Folge den Anweisungen während der Installation.
Lies bitte jede Anweisung genau. Gib, wenn gefragt, das Arbeitsverzeichnis (*working directory*) ein.



Arbeitsverzeichnis
Wir empfehlen das folgende Verzeichnis als Arbeitsverzeichnis:
C:\Program Files (x86)\maxon motor ag
Beachte, dass der Name des Programmverzeichnisses je nach verwendeter Systemsprache verschieden sein kann.

Schritt 3 Überprüfe die neuen Shortcuts und Symbole im Startmenu des Computers.
Die Dateien wurden ins Verzeichnis *maxon motor ag* kopiert, wo du Zugriff auf das Programm als auch die gesamte Dokumentation hast.
Klicken auf den *EPOS Studio* Shortcut auf dem Desktop startet das Programm.

Schritt 4 Falls nötig: Modifiziere oder entferne die Software.
Um Eigenschaften der Anwendung zu ändern oder das Programm zu entfernen, starte das Installationsprogramm *EPOS Positioning Controller.msi* nochmals und folge den Anweisungen.



Das neueste EPOS Studio
Stelle sicher, dass du die richtige Version installiert hast: EPOS Studio 2.00 (Revision 2) oder höher. Klicke im Menupunkt *Help* auf *About EPOS Studio*. Falls du nicht die neueste Ausgabe hast, lade die aktuellste Version von der maxon Website herunter http://www.maxonmotor.com.

Komponente	Minimale Anforderungen
Betriebssystem	Windows 8, Windows 7, Windows Vista, Windows XP SP3
Prozessor	Core2Duo 1.5 GHz
Laufwerk	Harddisk-Laufwerk, 1.5 GB verfügbarer Speicher DVD Laufwerk
Speicher	1 GB RAM
Bildschirm	Auflösung 1024 x 768 Pixels mit hoher Farbauflösung (16-Bit)
Web-Browser	Internet Explorer IE 7.0

Tabelle 1: Minimale Systemanforderungen (Vgl. Release Notes.txt *im EPOS Studio Installationsverzeichnis)*

1.2 Kabelanschlüsse

Die Verkabelung ist einfach und sollte ohne Fehler machbar sein. Verbinde zuerst den Motor (3 Kabel: rot, schwarz, weiss mit einem weissen Stecker) mit der einzigen passenden weissen Dose. Dann dasselbe mit dem Hallsensor-Kabel; seine Steckdose ist gleich neben den Motorleitungen. (Steck es nicht in die Dose die mit RS232 angeschrieben ist). Auch das Encoder-Flachbandkabel sollte kein Problem bieten.

Als nächstes verbinde die Leiterplatte mit der *EPOS2 P* mit Hilfe des kurzen Kabels mit den breiten Steckern. Auch dies geht einfach, die Stecker passen nur an einer Stelle. Dann – und gute Ingenieure tun dies am Schluss – verbinde das Netzgerät. Zuletzt stecke das USB Kabel in einen der USB Ausgänge deines Computers und in den *EPOS2 P* Mini USB.

Das Ganze sollte dann etwa wie in Abbildung 2 aussehen.

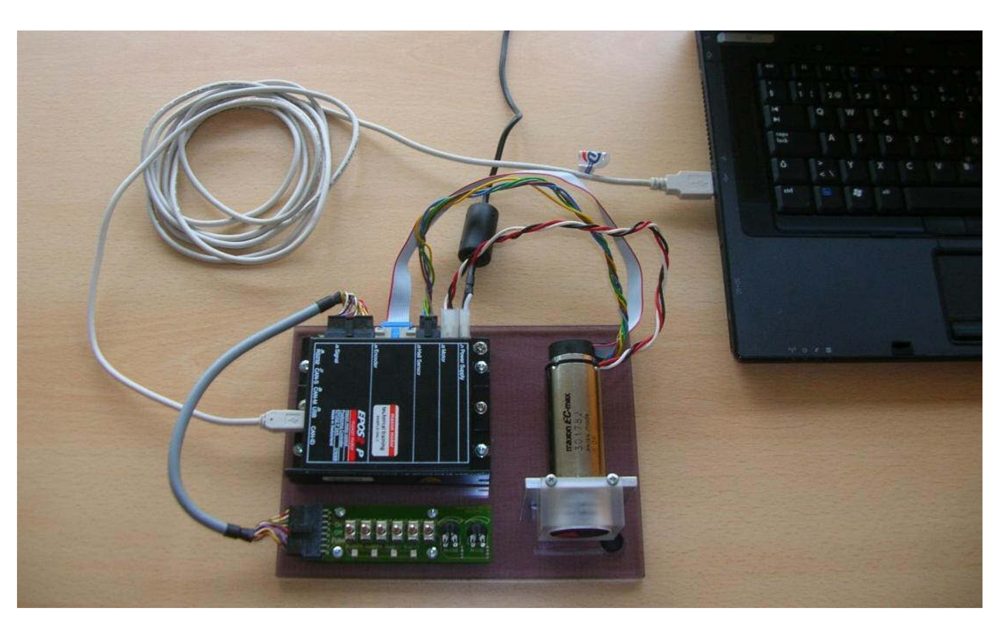


Abbildung 2: Wie das EPOS2 P Starter Kit verkabelt wird.



Den USB-Treiber finden
Beim ersten Anschliessen der EPOS2 P am USB-Port des Computers erscheint die Meldung, dass ein neues Gerät gefunden wurde und verlangt nach dem entsprechenden Treiber. Lass nicht den Computer automatisch nach dem Treiber suchen. Es geht schneller, wenn du manuell sagst, wo die Treiber zu finden sind. (Manchmal ist es nötig, zuerst den *DriverPreInstaller.exe* laufen zu lassen.)
Die Treiber befinden sich im selben Verzeichnis, wo die EPOS2 P Software installiert ist: *../maxon motor ag/ Driver Packages/ EPOS2 USB Driver* (siehe *Abbildung 4*). Folge den Schritten im Dokument *EPOS USB Driver Installation.pdf*. Der Ablauf kann je nach Betriebssystem leicht variieren.

1.3 Was befindet sich im schwarzen Gehäuse?

Um das schwarze Kästchen mit dem Namen *EPOS2 P* geht es in diesem Lehrbuch. Es ist ein maxon Motion Controller mit einer eingebauten Speicherprogrammierbaren Steuerung (SPS). EPOS steht für ***E****infach zu bedienendes* ***Po****sitionier-****S****ystem*, die **2** bezeichnet die zweite Generation, und **P** steht dafür, dass es programmierbar ist und eine SPS enthält. Allerdings ist es dir überlassen zu beurteilen, ob es wirklich "EPOS" ist oder nur "POS", d.h. ob es einfach zu bedienen ist oder nicht.

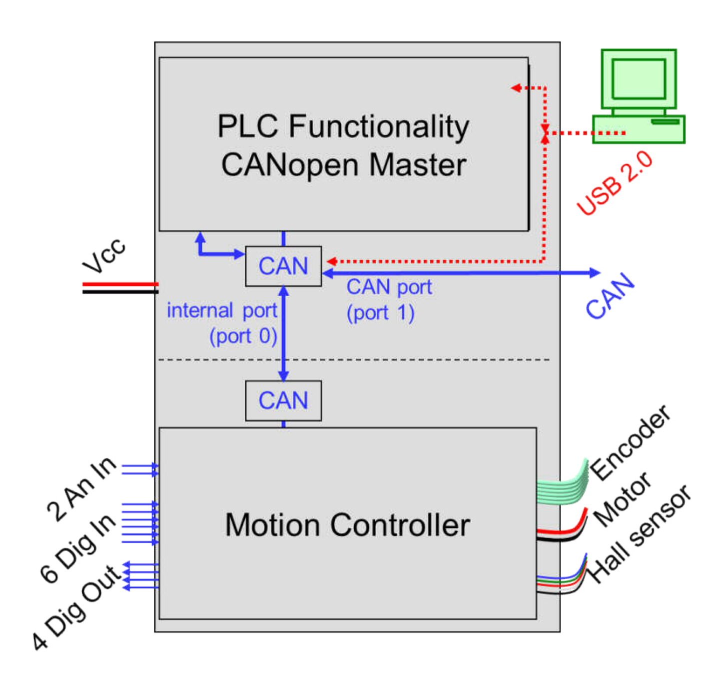


Abbildung 3: Schematische Übersicht der EPOS2 P
mit externen Anschlüssen und Kommunikation.

Im Endeffekt enthält die *EPOS2 P* zwei Geräte, eine SPS und einen Motion Controller, die intern über einen CAN-Bus verbunden sind. Das *EPOS Studio* auf deinem Computer kommuniziert mit diesen beiden Geräten über eine USB2.0 serielle Schnittstelle. USB2.0 ist dabei die voreingestellte Kommunikation. Alternative Möglichkeiten sind RS232 oder CAN-Bus. Da dein Computer höchstwahrscheinlich keine CAN-Schnittstelle hat, können wir diesen Bus nicht verwenden.

Der SPS-Teil ist dabei der Master des Gerätes. Er kontrolliert den Prozessablauf und alle Slaves im Netzwerk; einer der Slaves ist dabei der eingebaute Motion Controller. Die SPS führt ein anwendungsspezifisches Programm aus, das wir im Teil 3 dieses Lehrbuches genauer betrachten (beginnend mit Kapitel 7).



Betriebsanleitungen und Software-Dokumentation
Die Installation richtet nicht nur die *EPOS Studio* Software auf deinem Computer ein, sondern kopiert auch alle Betriebsanleitungen und die Software, die zur Programmierung des vorliegenden oder anderer maxon EPOS Geräte nötig ist.

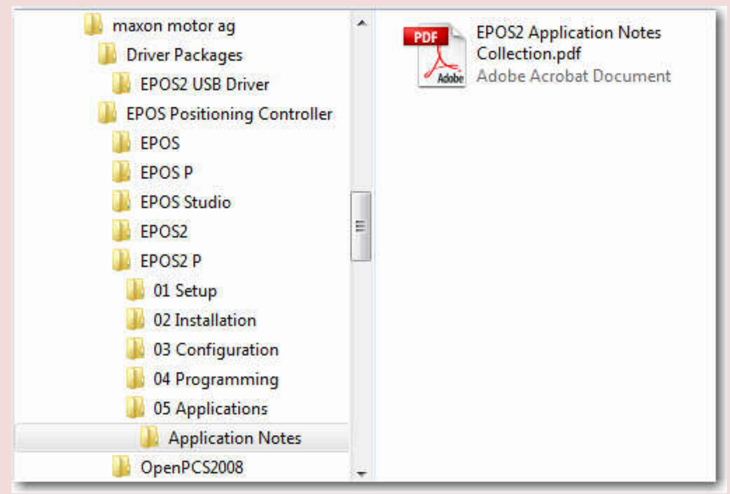


Abbildung 4: Die Dateistruktur der maxon EPOS Softwareinstallation.

Zu jedem EPOS Produkt gibt es 5 Unterverzeichnisse mit den Dokumenten wie in *Abbildung 5* gezeigt:

- In *01 Setup* befindet sich die *Getting Started* Anleitung mit im Wesentlichen demselben Inhalt wie dieses erste Kapitel. Zusätzlich findest du darin Verkabelungsinformationen für andere Motor-Encoder-Kombinationen.
- In *02 Installation* ist die Hardware-Verkabelung aufgeführt.
- *03 Configuration* enthält die aktuellen und älteren *Firmware*-Dateien; das ist die Software, die auf den Geräten läuft.
- *04 Programming* enthält wichtige Dokumente für die Programmierung: Die *Firmware Specification* (vgl. Kapitel 6.3). Dort sind alle Eigenschaften und Parameter im Detail beschrieben. Der Programmierer findet weitere nützliche Informationen, Programmierbeispiele und Bibliotheken im Dokument *Programming Reference* (vgl. Kapitel 7ff).
- *05 Application*: Die *Application Notes Collection* enthält Informationen zu besonderen Anwendungsbedingungen und Betriebsmodi.

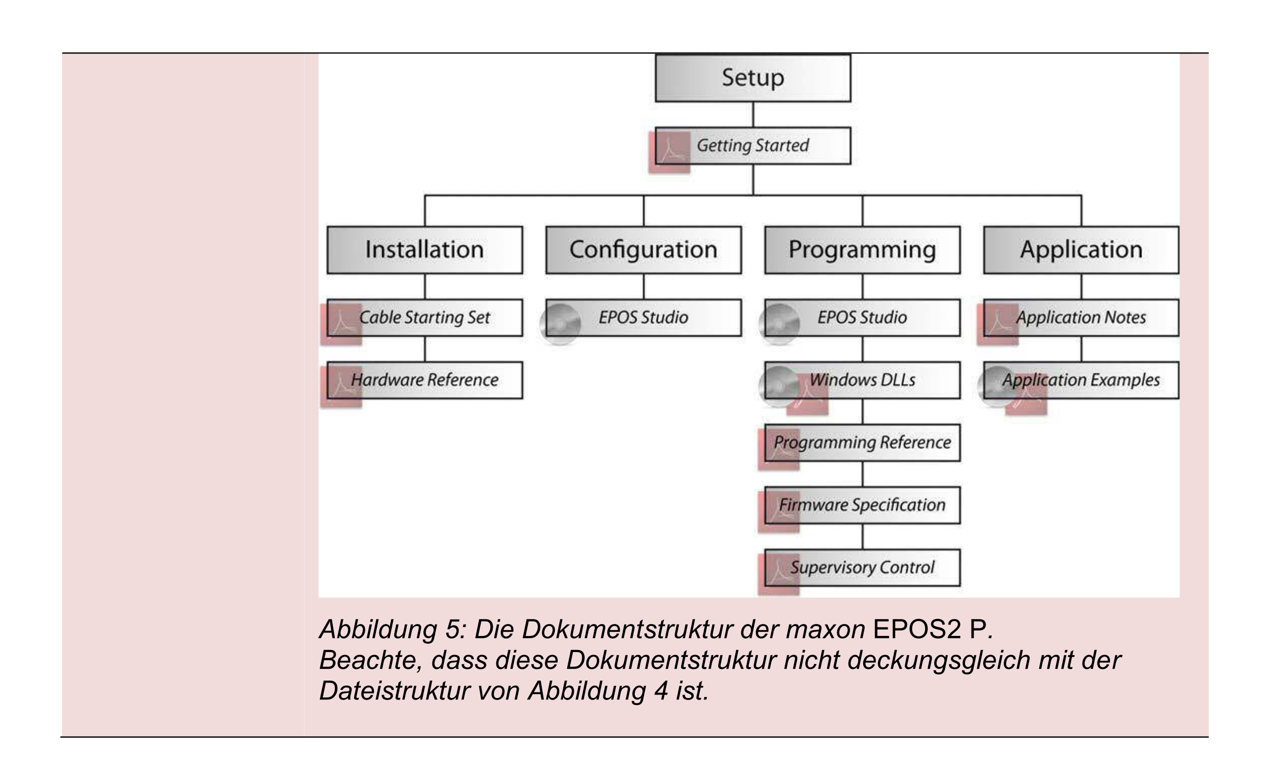


Abbildung 5: Die Dokumentstruktur der maxon EPOS2 P*.*
Beachte, dass diese Dokumentstruktur nicht deckungsgleich mit der Dateistruktur von Abbildung 4 ist.

1.4 Das EPOS Studio starten

Wir sind bereit! Starte das *EPOS Studio* durch Anklicken des entsprechenden Symbols auf deinem Computerdesktop.

Das *EPOS Studio* ist die graphische Benutzeroberfläche für alle maxon EPOS Produkte. Um eine Kommunikation mit dem aktuellen Gerät aufzubauen, muss das richtige EPOS Projekt angewählt werden. Der *New Project Wizard* öffnet sich automatisch. Falls nicht, klicke auf das *New Project* Symbol in der Menuleiste.

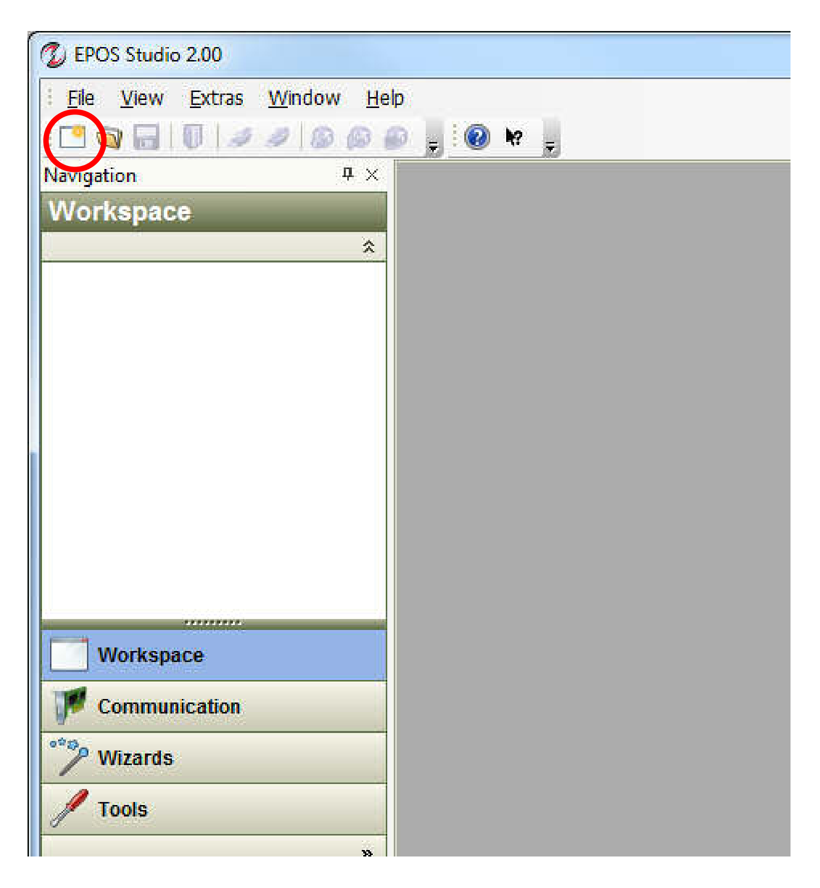


Abbildung 6: Wo sich das New Project *Symbol befindet.*



Projekt im EPOS Studio
Das *EPOS Studio* ermöglicht es, die Einstellungen deines Projekts zu speichern und es wieder zu öffnen. Das Projekt im *EPOS Studio* enthält Information über die Hardware und den benützten Kommunikationskanal. Falls mehrere EPOS in einem Netzwerk angeschlossen sind, so wird dies ebenfalls gespeichert.
Projekte unter einem eigenen Namen zu speichern ist immer dann nützlich, wenn du nicht ein Standardprojekt wie in diesem Fall hier verwendest.



Schritt 1 Wähle *EPOS2 P Project* aus der Liste. Klicke *Next (Weiter)*.

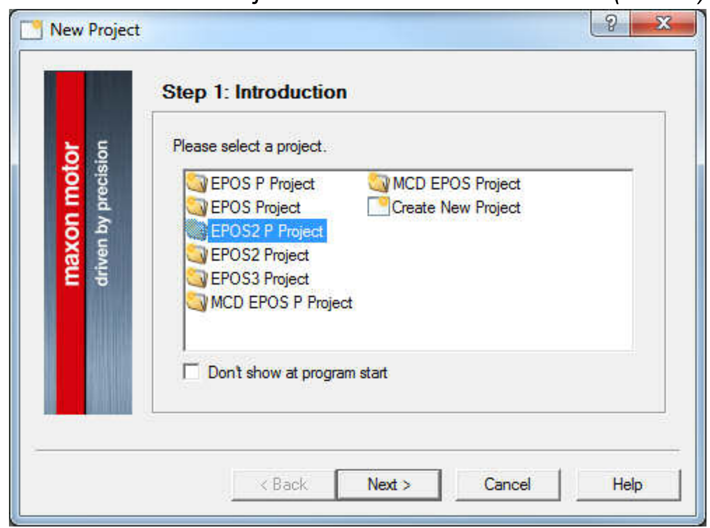


Abbildung 7: New Project Wizard, Schritt 1.

Schritt 2 Definiere Pfad und Namen für dein Projekt oder verwende die vorgeschlagenen.
Klicke *Finish (Beenden),* um ein neues Projekt zu erzeugen.

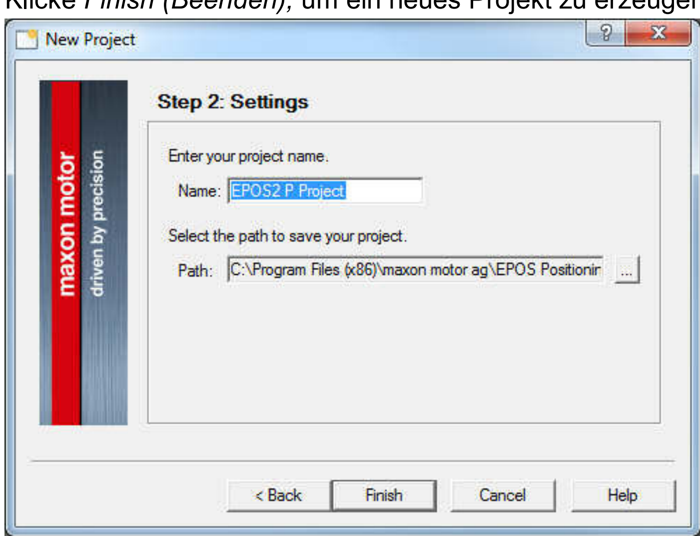


Abbildung 8: New Project Wizard, Schritt 2.

Wenn alles gut läuft, stellt das *EPOS Studio* eine Verbindung mit der *EPOS2 P* her. Dies erkennt man an den Balken, die von links nach rechts in pop-up Fenstern erscheinen und anzeigen, dass die Parameter der *EPOS2 P* gelesen werden.

Der *Workspace Tab*

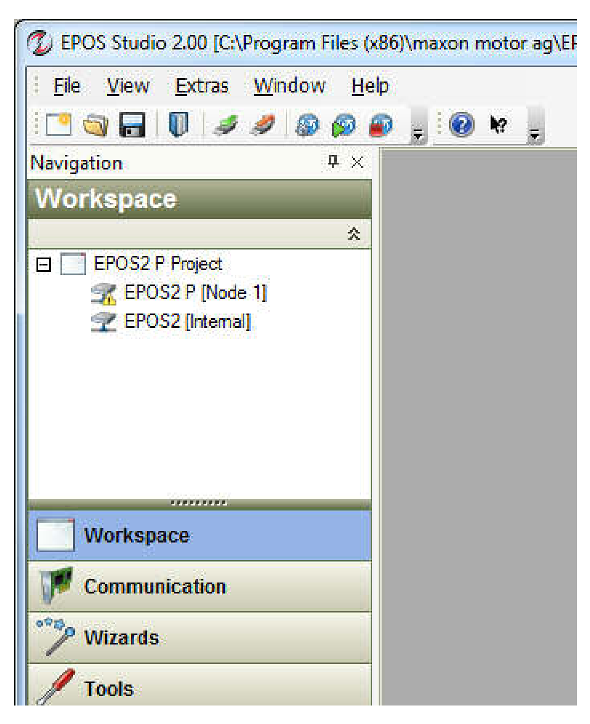
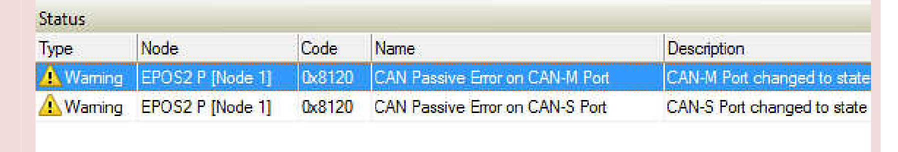


Abbildung 9: Der EPOS Studio *Bildschirm mit dem geöffneten* Workspace Tab.

Die Hardware des Projekts ist im *Workspace* des *Navigation* Fensters links dargestellt. In unserem Fall enthält das Projekt die *EPOS2 P [Node 1]*, was für das schwarze Kästchen als Ganzes steht oder spezifischer für die SPS darin, und die *EPOS2 [internal]*, welches den Motion Controller Teil repräsentiert.



Fehler (Errors) und Warnungen (Warnings)
Da keine CAN-Kommunikation eingerichtet ist, erscheinen die Warnhinweise *CanPassiveError on CAN-S Port* und *CanPassiveError on CAN-M Port*. Lösche die Warnungen mittels rechten Mausklicks auf die *Status*-Liste und wähle *Clear All Entries*.
Falls weitere Fehlermeldungen und Warnungen erscheinen, überprüfe die Verkabelung und die Konfiguration. (Weiteres zu Fehlern und Warnungen im separaten Dokument *EPOS2 Firmware Specification*).



Status

Type	Node	Code	Name	Description
Warning	EPOS2 P [Node 1]	0x8120	CAN Passive Error on CAN-M Port	CAN-M Port changed to state
Warning	EPOS2 P [Node 1]	0x8120	CAN Passive Error on CAN-S Port	CAN-S Port changed to state

Abbildung 10: Liste der Fehler und Warnungen im Status *Fenster.*

Der *Communication Tab*

Der *Communication Tab* beschreibt dein Projekt aus Kommunikationssicht. Auf dem Schema erkennst du deinen Computer (*Local host*), der via USB2.0 mit der *EPOS2 P [Node 1]* verbunden ist

Ein interner CAN-Bus verbindet die *EPOS2 P [Node 1]* mit dem eingebauten Motion Controller, der *EPOS2 [internal]*. Dieses Netzwerk heisst *CAN-I,* was für internes CAN Netzwerk steht. Man erkennt eine weitere CAN Verbindung ausgehend von der *EPOS2 P [Node 1]*, *CAN-S* genannt (für CAN Slave Netzwerk); allerdings ist da kein Gerät angeschlossen.

Das Schöne am *EPOS Studio* ist, das damit direkt mit allen Geräten im Netzwerk kommuniziert werden kann, ohne dass man etwas programmieren muss. Dies ist nützlich um das System einzurichten, für Schulungen oder irgendwelche andere Dinge, die man ausprobieren will. Im weiteren Verlauf dieses Lehrbuches benützen wir diese Kommunikation, zuerst, indem wir den internen *EPOS2* Motion Controller direkt ansprechen und später zur Programmierung der SPS in der *EPOS2 P*.

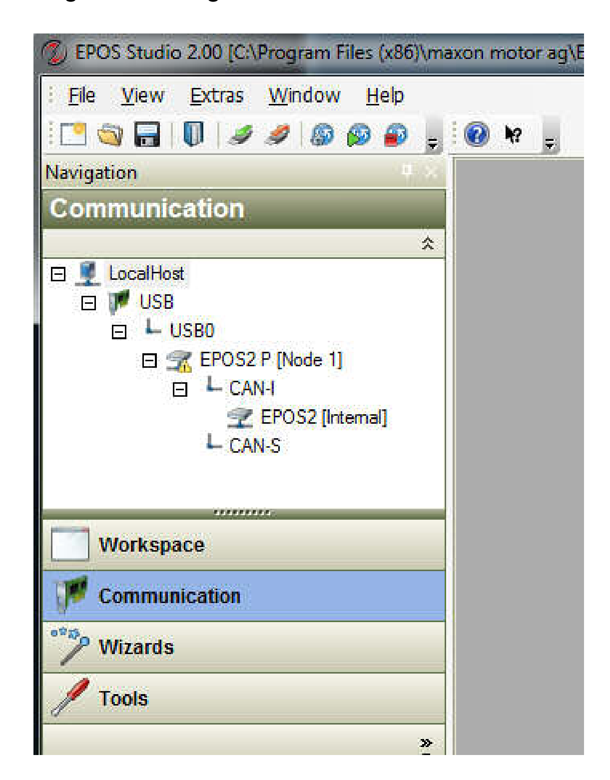


Abbildung 11: Der Communication Tab *auf dem* EPOS Studio *Bildschirm.*

Wizards Tab und *Tools Tab*

Im Moment brauchen wir *Wizards* und *Tools* nicht im Detail zu betrachten. Vieles wird später benötigt und, wenn es unsere Wege kreuzt, erklärt.

Nur eine Bemerkung: *Tools* und *Wizards* sind den jeweiligen Hardware-Komponenten des Projekts zugeordnet. Deshalb muss immer zuerst das korrekte Gerät aus der Auswahlliste gewählt werden. Wie du nun schon wissen solltest, haben wir zwei Geräte vor uns: Die *EPOS2 P* mit der SPS und der *EPOS2 [internal]* Motion Controller.

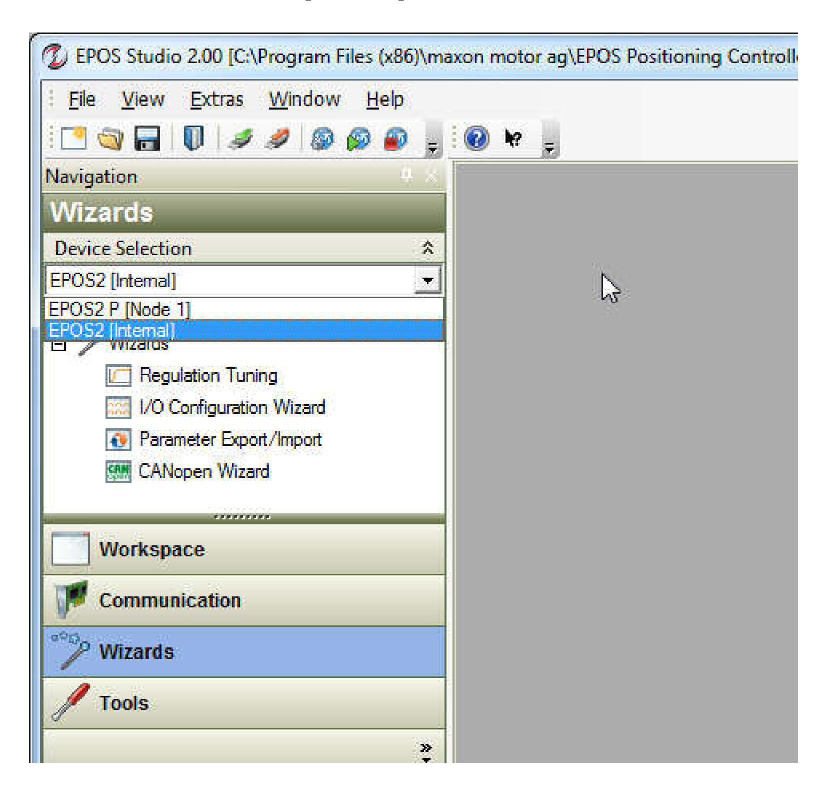


Abbildung 12: Das richtige Gerät auswählen im Wizards Tab.

2 Das Motion Control System vorbereiten

2.1 Startup Wizard

Eine *EPOS2 P* kann viele verschiedene Antriebskonfigurationen aus Motoren und möglichen Sensoren ansteuern, und – wie wir schon gelernt haben – mit mehreren Kommunikationsmöglichkeiten. Deshalb sollten wir zuerst die aktuelle Konfiguration definieren. Dazu benützen wir den *Startup Wizard*, den wir unter den *Wizards* der *EPOS2 P* finden. Doppelklicken auf das entsprechende Symbol reicht.

Glücklicherweise ist unsere Konfiguration ziemlich einfach. Nur wenige Parameter sind von den Standardeinstellungen verschieden und es gibt auch keine allzu grosse Auswahl an Möglichkeiten. Wenn du aber ein komplexeres System hast (z.B. eine *Dual-Loop-Konfiguration* mit zwei Feedback-Sensoren am Motor und an der Last), musst du dich sorgfältig durch den *Startup Wizard* durcharbeiten.

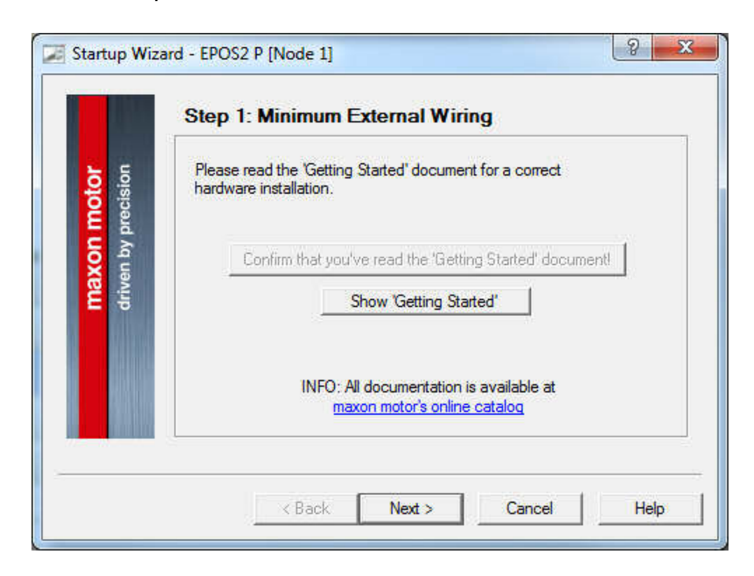


Abbildung 13: Startup Wizard*, Schritt 1.*

Schritt 1 Zuerst will maxon sicherstellen, dass du die richtige Dokumentation vor dir hast und weisst, was du tust. Darum musst du die entsprechende Schaltfläche (*Confirm that you've read the "Getting Started" document*) bestätigen. Kein Problem, wenn du das Ganze nicht gelesen hast. Im Grunde genommen tust es gerade jetzt. Diese ersten Kapitel enthalten dieselben Informationen wie das *Getting Started*. Also entspannen und *Next* klicken.

Schritt 2 Im zweiten Schritt gilt es den Kommunikationskanal festzulegen. USB ist als Default-Kommunikation schon festgelegt. Somit nur auf *Next* klicken. Wenn es dich interessiert, welche anderen Möglichkeiten es gibt, tue dir keinen Zwang an und schaue nach. Stelle nur sicher, dass die Einstellungen am Ende der folgenden Abbildung entsprechen.

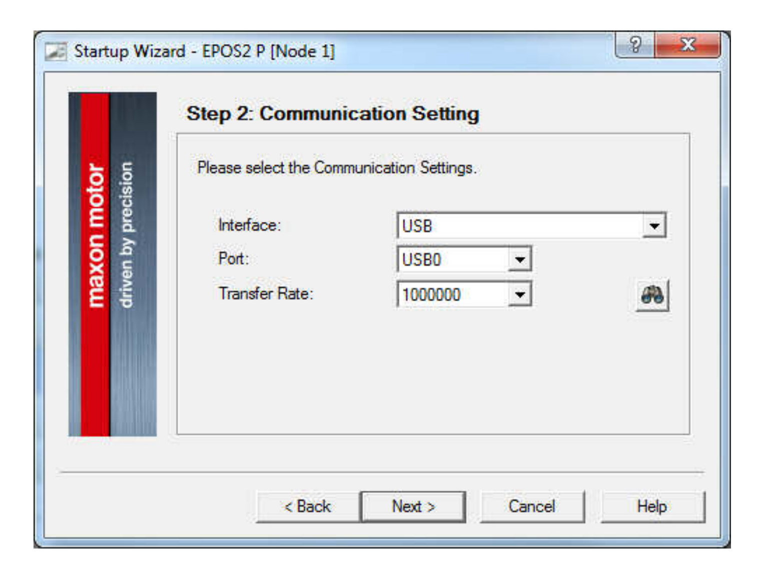


Abbildung 14: Startup Wizard*, Schritt 2.*

Schritt 3 Der dritte Schritt definiert den verwendeten Motortyp. Der Motion Controller kann mit bürstenbehafteten DC-Motoren (maxon DC motor) oder mit bürstenlosen DC-Motoren (maxon EC motor). Da unser Motor ein maxon EC motor ist, wähle die zweite Option. Dann wiederum *Next*.

Schritt 4 Bei bürstenlosen Motoren musst du nun die Kommutierungsart festlegen. Die beste Leistung erhält man mit Sinuskommutierung, welche allerdings einen Motor mit Encoder und Hallsensoren benötigt. Diese perfekten Voraussetzungen sind bei unserem Motor erfüllt. Somit wählen wir diese Kommutierung und klicken auf *Next*.

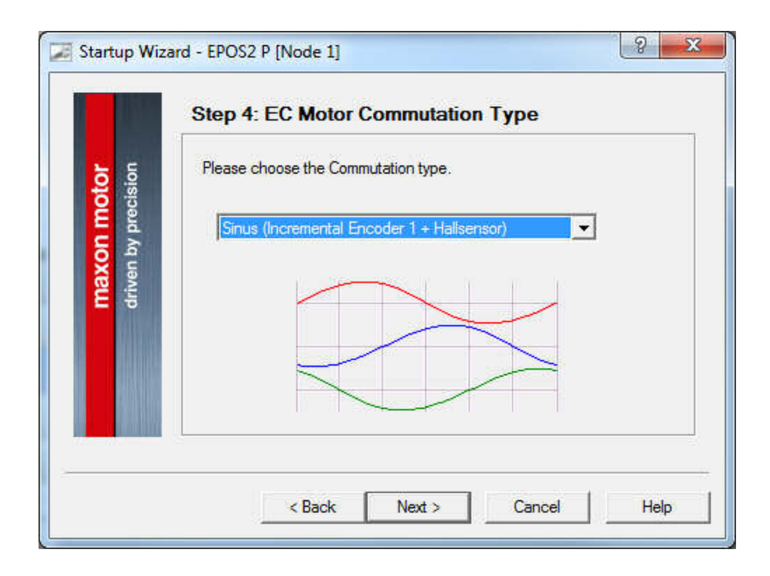


Abbildung 15: Startup Wizard*, Schritt 4.*

maxon EC Motor-typen (Beispiele)	Anzahl Polpaare	Max. Drehzahl bei Blockkommutierung	Max. Drehzahl bei Sinuskommutierung
maxon EC Motor, EC-max	1	100'000 rpm	25'000 rpm
EC-4pol	2	50'000 rpm	12'500 rpm
EC 20 flach, EC 32 flach	4	25'000 rpm	6'250 rpm
EC-i 40, EC 60 flach	7	14'280 rpm	3'570 rpm
EC 45 flach	8	12'500 rpm	3'125 rpm
EC 90 flach	12	8'330 rpm	2'080 rpm

Tabelle 2: Maximale Drehzahl von EPOS-Systemen mit bürstenlosen DC-Motoren als Funktion der Anzahl Polpaare und Kommutierungstyp. Beachte, dass viele Motoren eine tiefere Grenzdrehzahl haben, als was mit Blockkommutierung möglich ist.

Schritt 5 In Schritt 5 wird festgelegt, welchen Feedbacksensor für Positionierung oder Drehzahlmessung verwendet wird. Der Standardsensor für maxon EPOS Controller ist ein digitaler Inkrementalencoder. Wir können denselben Encoder verwenden, den wir im vorangehenden Schritt für die Kommutierung bestimmt haben. Somit wiederum nur *Next* klicken.

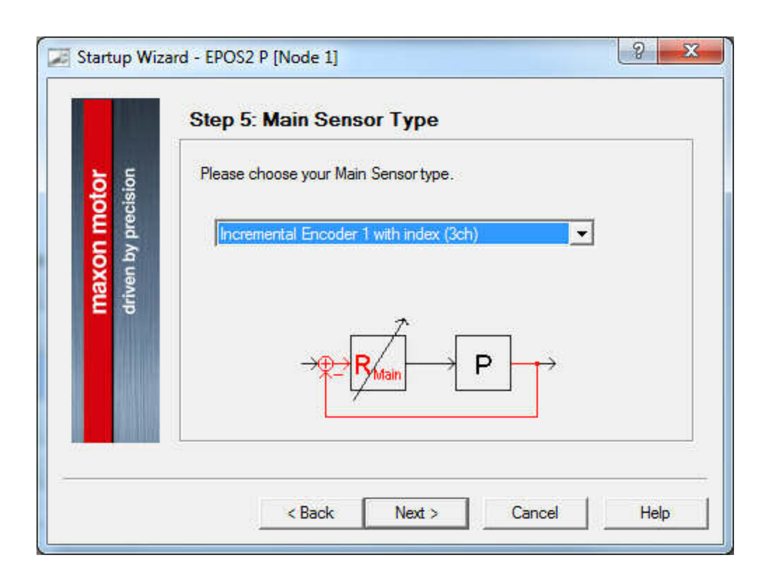


Abbildung 16: Startup Wizard*, Schritt 5.*

Schritt 6 Nun gilt es einige Parameter einzugeben, die gebraucht werden um einen sicheren Motorbetrieb zu gewährleisten. Diese Parameter sind auf dem Datenblatt des Motors angegeben (vgl. Kapitel 11.1, Motor-No. 272763).
Wir spezifizieren die *Grenzdrehzahl* (*Maximum permissible speed*) des Motors (15'000 rpm für den aktuellen Motor, oder auch ein kleinerer Wert, falls die Anwendung aus anderen Gründen eine Drehzahlbegrenzung verlangt).
Nennstrom (*Nominal current* = 2660 mA) und *Thermische Zeitkonstante der Wicklung* (*Thermal time constant of the winding* = 2.7 s) werden benötigt, um den Motor vor dem Überhitzen zu schützen. Der Maximalstrom (*Max. Output Current Limit*) wird automatisch auf den doppelten Nennstrom gesetzt. (Er kann später im *Object Dictionary* geändert werden; vgl. Kapitel 6.4)
Die Anzahl Polpaare (*Number of pole pairs*) - für unseren Motor ist der Wert 1 – wird zur korrekten Kommutierung und Drehzahlmessung gebraucht.

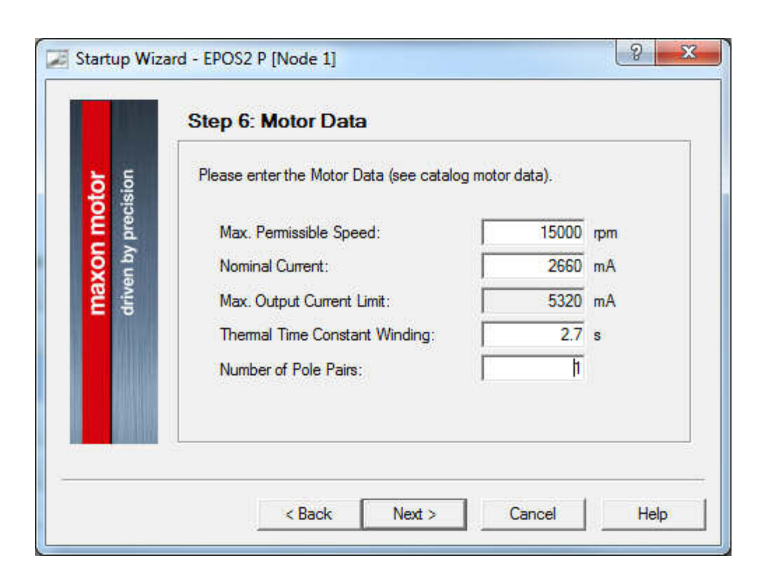


Abbildung 17: Startup Wizard*, Schritt 6.*

Schritt 7 Nach *Next* definieren wir die Eigenschaften des Encoders. Er hat 500 Impulse pro Umdrehung (*pulse/turn*). Und wiederum *Next*.

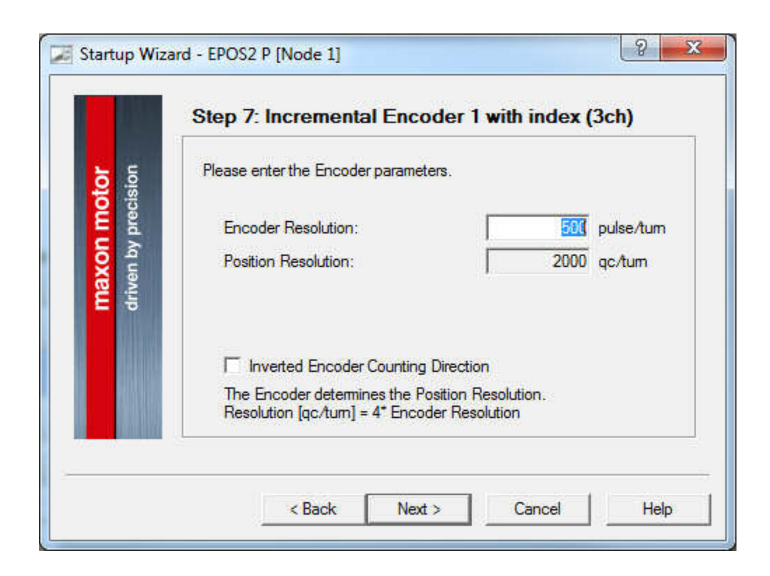


Abbildung 18: Startup Wizard*, Schritt 7.*

Schritt 8 Den maximalen Schleppfehler (*Maximum following error)* belassen wir bei den vorgeschlagenen 2000 qc (= quadrature counts, vgl. Hintergrund-information zu *Positions- und Drehzahlbestimmung mit Inkrementalencoder*) und wir gehen weiter zum letzten Fenster, das eine Zusammenfassung unserer Einstellungen gibt.
Beende (*Finish)* diesen Dialog und bejahe die Frage (*Yes*), ob diese Parameter und Einstellungen gespeichert werden sollen.

Das ist, was wir mit *einfach zu bedienen* meinen: Mit ein paar wenigen Klicks und Parameter ist das System eingerichtet.



Kommutierung von Motoren mit und ohne Bürsten
Die *Kommutierung* beschreibt, wie die Endstufe des Reglers die Versorgungsspannung an die verschiedenen Wicklungssegmente anlegt und weiterschaltet.

Bei Motoren mit Bürsten – z.B. maxon DC motor – erfolgt die Kommutierung im Motor mechanisch mit Hilfe von Graphit- oder Edelmetallbürsten. Damit der Motor dreht, muss nichts weiter getan werden als die Spannung an den beiden Motoranschlüssen anzulegen.
Bürstenbehaftete Motoren brauchen keinen zusätzlichen Sensor für die korrekte Kommutierung.

Bürstenlose DC Motoren (BLDC) – z.B. maxon EC motor – haben drei Wicklungssegmente (Phasen). Die Endstufe des Reglers muss die Versorgungsspannung korrekt an diese drei Phasen in Abstimmung mit der aktuellen Rotorposition anlegen. Für die einfachste Kommutierungsart, *Blockkommutierung* genannt, wird die Rotorposition mittels dreier digitaler Hallsensoren bestimmt. Kurz und ohne in die Details zu gehen, sind die Hallsensoren dazu da, den BLDC-Motor korrekt zu bestromen. Allerdings ergeben die Hallsensoren nur wenig Auflösung und die Blockkommutierung hat ihre Grenzen, wenn es um hohen Gleichlauf bei kleinen Drehzahlen geht.
Eine ausgereiftere Form der Kommutierung ist die *Sinuskommutierung* . Sie ergibt einen hohen Gleichlauf - sogar bei den tiefsten Drehzahlen - und sogar einen leicht höheren Motorwirkungsgrad. Allerdings benötigt sie zusätzlich die höhere Auflösung eines Encoders als Feedbacksensor. Dieser ist aber in Servosystemen mit Positions- oder Drehzahlregelung meist sowieso vorhanden.
Beachte: Bei bürstenlosen DC-Motoren wird das Feedback für zwei separate Aufgaben benötigt: Für die Kommutierung – d.h. für den Motorbetrieb – und für die Drehzahl- oder Positionsregelung.

Die EPOS2 Motion Controller erlauben Blockkommutierung für Drehzahlen bis 100'000 rpm, während Sinuskommutierung bis maximal 25'000 rpm möglich ist. Diese Werte gelten für BLDC-Motoren mit *einem* magnetischen Polpaar. Bei höheren Anzahl Polpaaren reduziert sich die Drehzahl entsprechend. Ein EC-4pole Motor mit 2 Polpaaren kann maximal noch bei der Hälfte dieser Drehzahlwerte betrieben werden. Flachmotoren haben noch mehr Polpaare und dem entsprechend eine tiefere Drehzahlgrenze.



Motor- und Encoderparameter der verwendeten Einheit
Die Bezeichnung des verwendeten Motors ist *EC-max 30*, mit der Bestellnummer 272763. Seine genauen Spezifikationen finden sich auf der maxon Website oder im Anhang (Kapitel 11). Für uns ist wichtig, dass es sich um einen bürstenlosen Motor (EC) mit Hallsensoren handelt.
Dieser Motor hat eine Grenzdrehzahl von 15 000 rpm, was viel höher ist, als was wir mit der vorhandenen Spannungsversorgung erreichen können (19 VDC). Typischerweise wird der Motor mit maximal etwa 6500 rpm bei 19 V Versorgung drehen (Drehzahlkonstante des Motors mal Motorspannung = 393 rpm/V * 0.9 * 19V). Somit besteht keine Gefahr, dass der Motor zu schnell dreht.
Der Nennstrom beträgt 2.66 A. Dies ist der Motorstrom, den der Motor dauernd erträgt, ohne zu überhitzen. Für einige Sekunden ist auch ein höherer Strom zulässig, wahrscheinlich bis etwa 10 A. Aber wir werden gar nicht so viel Strom aus unserem Netzgerät ziehen können, es kann weniger als 5 A liefern. Somit besteht auch kein Risiko, den Motor zu überhitzen.

Der Encoder ist ein *MR Encoder*, Bestellnummer 225778. Er hat 500 Impulse pro Umdrehung auf jedem der beiden Kanäle. Dies ergibt 2000 Signalflanken - quad counts (qc) genannt - pro Motorumdrehung. Qc ist die interne Einheit für die Position. Somit resultiert eine nominelle Auflösung von 1/2000 Umdrehung (oder 0.18°).
Zusätzlich hat dieser Encoder einen Indexkanal, der für präzise Referenzfahrt gebraucht werden kann und benützt wird, um die Hallsensor-Signale zu überprüfen.



Positions- und Drehzahlbestimmung mit Inkrementalencoder
Digitale Inkrementalencoder sind die am meisten verwendeten Sensoren um Drehzahl und Position in Kleinstantrieben zu messen. Sie können sich in Aufbau und Funktionsprinzip stark unterscheiden, aber im Grunde unterteilen sie alle eine Motorumdrehung in eine bestimmte Anzahl Schritte (Inkremente) und senden für jedes Inkrement einen Signalpuls zum Regler. Inkrementalencoder haben in der Regel Rechteckimpulse auf zwei Kanälen, A und B, die gegeneinander um eine Viertelsignallänge (oder 90 elektrische Grad) verschoben sind.

Die **charakteristische Grösse** des Encoders ist die Anzahl Impulse *N* pro Umdrehung, angegeben als cpt (*counts per turn*). Wenn man die Zustandsänderungen in beiden Kanälen berücksichtigt, ergibt sich eine viermal höhere Auflösung. Diese Zustände werden als *quad counts* (qc) bezeichnet und als Positionseinheit in EPOS Systemen verwendet.

Beachte: Wenn über die Auflösung von Inkrementalencoder gesprochen wird, soll man klären, ob man vor oder nach der Quadratur meint.
Je höher die Auflösung, umso genauer kann man die Position bestimmen und umso präziser kann die Drehzahl aus der Positionsänderung pro Zeit berechnet werden.

Die **Drehrichtung** folgt aus der Abfolge der Signalpulse von Kanal A und B: Kanal A führt in eine Drehrichtung, Kanal B in die andere.

Weiter kann der Encoder einen **Indexkanal**aufweisen. Dies ist ein einzelner Impuls pro Umdrehung. Der Index kann zur genauen Bestimmung der Referenzposition verwendet werden (vgl. Homing, Kapitel 3.2). Einige Regler benützen den Index zusätzlich als absolute Positionsreferenz bei der Sinuskommutierung von bürstenlosen Motoren.

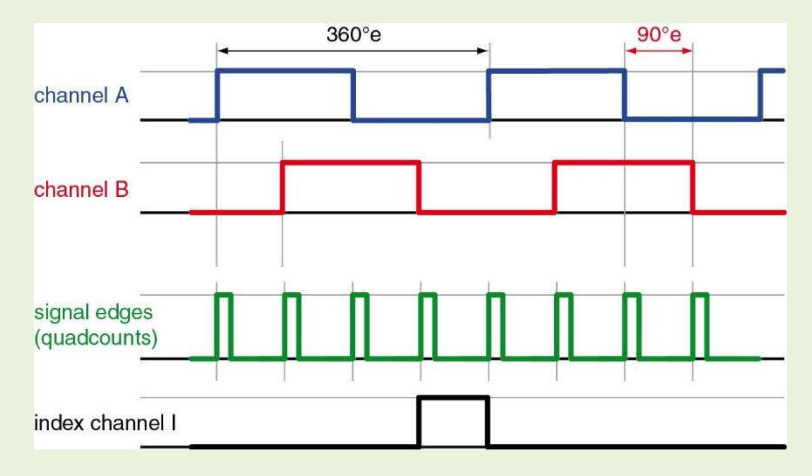


Abbildung 19: Die Signale eines Inkrementalencoders.

Die **Position** wird durch zählen der Signalflanken (Inkremente) von einer definierten Referenzposition (vgl. Homing in Kapitel 3.2) aus bestimmt. Die Einheit der Position ist *quad counts* (qc).

Die **Drehzahl** wird aus der Änderung der Position pro Abtastintervall berechnet. In unserem Fall ist die natürliche Einheit der Drehzahl quad counts pro Millisekunde. Somit können Drehzahlen nur in Schritten von 1 qc/ms berechnet werden. Üblicherweise wird dies in die praktischere Einheit rpm umgerechnet und angezeigt. Die Auswirkung dieser Quantisierung der Drehzahl eines Inkrementalencoders erkennt man sehr schön auf den gemessenen Drehzahldiagrammen (vgl. Kapitel 3.4).

Beschleunigungs- und Bremswerte werden in der praktischen Einheit rpm/s angegeben.



Encoderimpulszahl und Regelungsdynamik
Eine höhere Encoderimpulszahl *N* ergibt nicht nur eine bessere Positionsauflösung, sondern verbessert auch die Regelungscharakteristik. Feedback-Informationen über Positionsänderungen erfolgen schneller und der Regelkreis kann schneller korrigieren. Damit kann der Regler steifer eingestellt werden, d.h. mit höherer Verstärkung, was das Risiko von instabilen Oszillationen vermindert.
Bei tiefer Impulszahl und hohen Werten der Regelparameter hat der Antrieb mehr Zeit zu beschleunigen, bevor eine Rückmeldung einer Positionsänderung (d.h. ein Encoderimpuls) auftritt. In diesen Fällen kann die Endposition überschossen werden und eine Korrektur in die Gegenrichtung wird eingeleitet. Dies kann wiederum in ein Überschiessen münden, und so weiter. Die Bewegung wird instabil und der Antrieb oszilliert.

2.2 Tuning

Der nächste Schritt zur Systemvorbereitung ist das Tuning. Der Zweck ist derselbe wie beim Tuning eines Rennwagens: Wir möchten eine schnelle, starke und optimierte Reaktion des Antriebssystems auf alle Bewegungsbefehle. Wir benützen dazu wiederum einen Wizard, der uns die Arbeit erleichtert.

Tuning hat mit dem Motion Control Teil unseres schwarzen Kästchens zu tun. Wähle deshalb den Motion Controller, *EPOS2 [internal]*, als aktuelles Gerät im *Wizards* Tab. Starte dann mit Doppelklick den *Regulation Tuning* Wizard.

Wähle die einfachere Option, das *Auto Tuning*, und dann weiter mit *Next*.

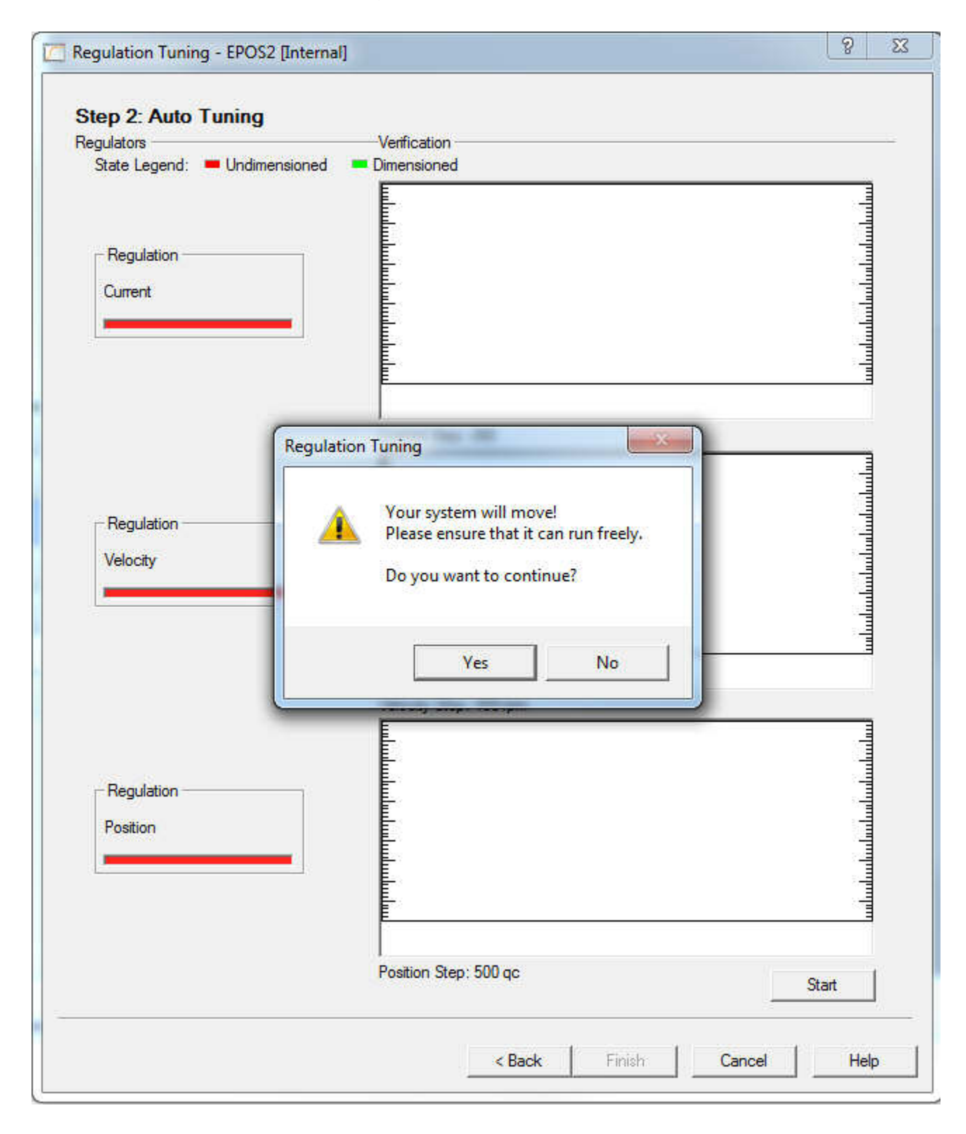


Abbildung 20: Regulation Tuning Wizard*: der* Auto Tuning *Bildschirm.*



Expert Tuning und manuelles Tuning
Die Wahl von *Expert Tuning* beim Starten des *Regulation Tuning Wizard* ermöglicht einen grösseren Einfluss auf den Tuningprozess. Aber zuerst musst du den Hauptregelparameter (*Main Control Parameter*) des Systems festlegen: Position, Geschwindigkeit (V*elocity*) oder Strom (C*urrent*).

Zuoberst auf dem Dialogfenster hast du Zugriff auf die Tuningparametern jedes Regelkreises. Du kannst die Werte von Hand ändern, wenn du mit dem Resultat des Tuning Wizard nicht zufrieden bist; dies ist dann das manuelle Tuning.

Als nächstes kommt der Abschnitt *Identification*, wo du die Amplitude der Oszillationsbewegung einstellen kannst. Ist die System-Identifikation abgeschlossen, sind das Systemverhalten, sowie die Effekte von Reibung und Massenträgheiten bekannt. Deshalb ist es auch unnötig, die Identifikation zu wiederholen, wenn man nur die Einstellungen im Abschnitt Parametrierung (*Parameterization*) ändert. Und sicher willst du die Identifikation weglassen, wenn du manuell tunest.

Die Art und Weise, wie die aktuellen Tuningparameter im *Parameterization* Schritt berechnet werden, hängt vom gewünschten Systemverhalten und der Anwendung ab. Vielleich möchtest Du ein möglichst steifes und schnell reagierendes System, und ein Überschwingen der Endposition oder der Zielgeschwindigkeit spielt keine Rolle. Dann stelle die *Regulation Stiffness* auf *hard*.
Oder du stellst eine weiche (*soft*) *Regulation stiffness* ein mit langsamerer Reaktion und weniger Überschwingen.
Versuche verschiedene Einstellungen, beobachte die Resultate in den *Verification* Diagrammen und vergleiche die Parameterwerte. Reduziere die *Max. Recording Time*, damit du die Unterschiede besser siehst; in unserem Fall sind 200ms oder sogar noch kürzer perfekt.

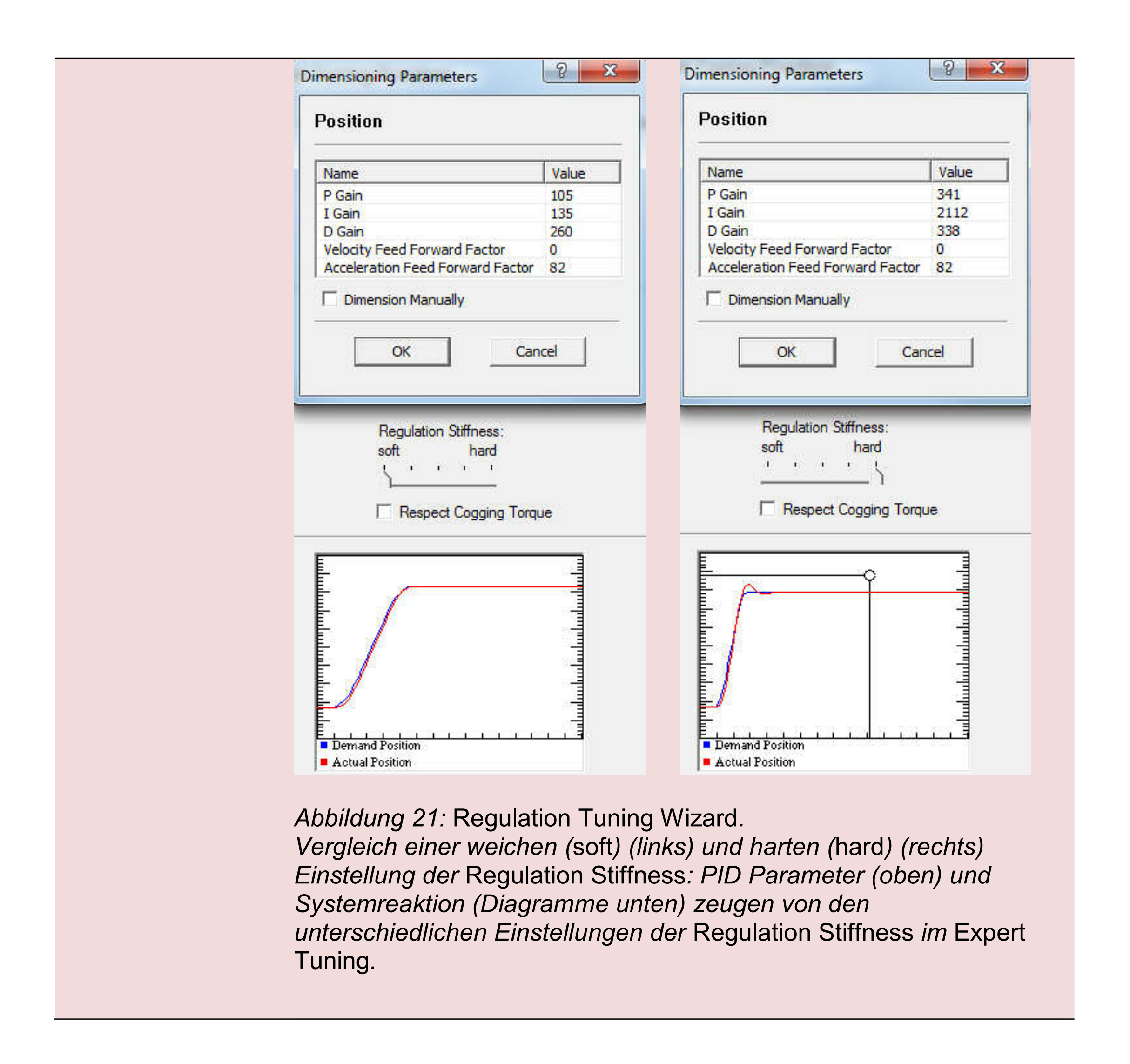


Abbildung 21: Regulation Tuning Wizard*.*
*Vergleich einer weichen (*soft*) (links) und harten (*hard*) (rechts) Einstellung der* Regulation Stiffness*: PID Parameter (oben) und Systemreaktion (Diagramme unten) zeugen von den unterschiedlichen Einstellungen der* Regulation Stiffness *im* Expert Tuning*.*

Starte das Tuning und bestätige die nächste Mitteilung, dass sich das System bewegen wird, mit Ja (*Yes*). Viel mehr bleibt nicht zu tun. Ein automatisches Tuning wird durchgeführt, beginnend im Stromregelkreis mit dem Optimieren der Antwort des Motorstroms. Das Ziel ist, den Strom so schnell und genau wie möglich zu regeln aufgrund der Bedürfnisse des übergeordneten Positions- oder Geschwindigkeitsregelkreises. Ein gut eingestellter Stromregelkreis ist eine Voraussetzung für eine dynamische Systemreaktion.

Das Autotuning lässt den Motor mit steigender Amplitude bei zwei verschiedenen Frequenzen oszillieren. Diese *Identifikation*, dient dazu, Information über das dynamische Verhalten des Systems zu erhalten.
Im nächsten Schritt des Tuning, der *Parametrisierung*, werden die optimalen Feedback- und Feed-forward-Parameter berechnet. Dies geschieht mittels der Resultat der Identifikation und basierend auf einem allgemeinen Modell des Antriebssystems.
Die V*erifikation* am Ende des Autotuning führt eine Bewegung aus, zeichnet den Strom-, den Geschwindigkeits- und den Positionsverlauf auf und stellt diese in den Diagrammen dar.



Diagramm-Zoom
Du kannst in die Diagramme zoomen. Wähle mit der Maus den interessierenden Bereich aus.
Mit rechtem Mausklick zoomst du eine Stufe zurück.

Speichere die neu gefundenen Tuningparameter permanent, wenn du den *Regulations Tuning Wizard* verlässt. Die Parameter gelten nun für alle zukünftigen Bewegungen. Sie sind permanent gespeichert und bleiben auch nach einem Neustart des Systems aktiv. Der Autotuning-Prozess liefert gute Resultate für die meisten Anwendungen; und dies nur mit einem Mausklick. Man braucht nicht ein mühseliges Versuch-Irrtum-Prozedere oder eine zeitaufwändige Analyse des aufgezeichneten Antwortverhaltens. Und dies, denken wir, läuft wiederum unter *einfach zu bedienen*.

Bemerkung: Tuning sollte immer am vollständigen System durchgeführt werden mit der tatsächlich eingesetzten Spannungsversorgung und mit allen mechanischen Komponenten, Massenträgheiten und Reibeffekten. Im Endeffekt wollen wir ja, dass die gesamte Maschine optimal reagiert und nicht nur der Motor alleine.



Feedback und Feed-Forward (Vorsteuerung)

PID Feedback-Verstärkung

PID steht für Proportionale, Integrale und Derivative Regelparameter. Sie beschreiben, wie das Fehlersignal *e* (siehe Abbildung 22) verstärkt wird, um eine geeignete Korrektur zu erhalten. Ziel ist es, diesen Fehler – d.h. die Abweichung zwischen Sollwert (*demand value*) und gemessenem Wert (*actual value*) zu verkleinern. Kleine Werte der Regelparameter ergeben meist ein weiches Regelverhalten. Hohe Werte ergeben eine steifere Regelung mit dem Risiko des Überschiessens und bei zu hoher Verstärkung kann das System zu oszillieren beginnen. Das kann man leicht selber ausprobieren, indem man die Autotuning-Parameter von Hand um etwa einen Faktor 5 erhöht.

Bei Drehzahlreglern wird der Regelkreis gewöhnlich mit einem einfachen PI-Algorithmus implementiert. Für Positioniersysteme ist ein derivativer Term zusätzlich nötig. Die drei Terme können sich wechselseitig beeinflussen und das Verständnis für diese Interaktion ist besonders für das Feintuning von Positioniersystemen wichtig. Für eine optimale Systemleistung müssen die Koeffizienten K_P, K_I und K_D je nach vorgegebener Bewegung und Lastträgheit neu eingestellt werden. (Literatur: Feinmess).

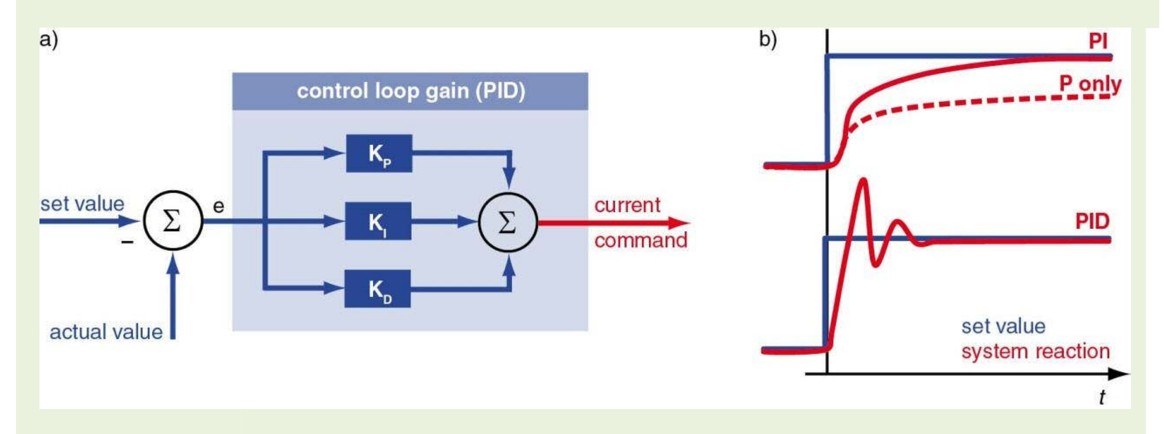


Abbildung 22: PID Controller.
a) Schematische Darstellung eines Positions- oder Drehzahlreglers mit einem PID-Element zur Verstärkung des Fehlersignals e.
b) Systemantwort bei einen Sollwertsprung für ein P, ein PI oder ein PID-Element.

Proportional-Regler (P): Die Regelabweichung *e* (Differenz zwischen Ist- und Sollwert) wird mit einem vom Anwender angegebenen Faktor K_P multipliziert und dann als neuer Strombefehl gesendet. Ein erhöhter K_P -Wert beschleunigt die Fehlerkorrektur. Ist K_P jedoch zu groß, tritt eine starke Überschwingung auf, und ab einem bestimmten Punkt kann das System in Schwingung versetzt werden, was bei unzureichender Dämpfung zu Instabilität führt.
K_P kann die Regelabweichung *e* nicht ganz beseitigen, da die proportionale Korrekturgröße ($K_P \cdot e$) mit abnehmender Regelabweichung *e* ebenfalls kleiner wird. Das Ergebnis ist eine bleibende Regelabweichung, die vor allem in Systemen bedeutend ist, die schon zur Aufrechterhaltung der Bewegung (z.B. aufgrund einer hohen Reibung) einen grossen Strombedarf haben.

Integral-Regler (I): Hier wird die Regelabweichung über längere Zeit summiert, mit einem vom Anwender angegebenen Faktor K_I multipliziert und zum neuen Strombefehl addiert. Diese Methode vermeidet eine bleibende Regelabweichung einer rein proportionalen Verstärkung, da andauernde Fehler eine verstärkte Korrektur auslösen.
Diese Methode hat aber einen Nachteil, vor allem wenn der Fehler zwischen positiven und negativen Werten schwankt. Da sich vergangene Fehler erst zeitverzögert auswirken, kann es zu positivem Feedback kommen, was zu einer Destabilisierung des gesamten Regelkreises führt. Hohe K_I –Werte könnten ohne eine entsprechende Dämpfung starke Systemschwingungen verursachen.

Differenzial-Regler (D): Der D-Regler berücksichtigt die Änderung der Regelabweichung, welche mit einem vom Anwender angegebenen Faktor K_D multipliziert dem Strombefehl hinzugefügt wird. Plötzlich auftretende Regelabweichungen, wie zum Beispiel nach einem Sollwertsprung, können damit sehr schnell korrigiert werden.
Richtig eingestellt kann mit dieser Art von Regelung die Stabilität verbessert werden. Sie kann als eine Art elektronische Dämpfung betrachtet werden. Eine Erhöhung des K_D-Wertes führt zu einer höheren Systemstabilität. Eine bleibende Regelabweichung wird jedoch nicht beseitigt, da die Ableitung einer Konstanten gleich Null ist.

Feed-Forward, Vorsteuerung
Bei den PID-Algorithmen gibt es nur dann ein Korrektursignal, wenn eine Regelabweichung existiert. Das bedeutet bei Positioniersystemen, dass es während einer Bewegung immer einen Positions-Folge-Fehler (einen Schleppfehler, *following error*) gibt, ja geben muss. Ziel der Feed-Forward-Steuerung ist die Minimierung dieses Schleppfehlers, indem das zukünftige Systemverhalten abgeschätzt und dem Regelkreis im Voraus die zu erwartende Regelabweichung mitgeteilt wird. Dazu stehen im Allgemeinen zwei Korrekturgrössen zur Verfügung, die ebenfalls wieder spezifisch für die Anwendung und die Bewegungsaufgabe bestimmt werden müssen:

- Geschwindigkeits-Feed-Forward-Verstärkungsfaktor: Er wird mit der erwünschten Geschwindigkeit multipliziert und kompensiert eine geschwindigkeitsproportionale Reibung.
- Beschleunigungs-Feed-Forward-Korrektur: Er berücksichtigt die Massenträgheit und stellt genügend Strom zu deren Beschleunigung bereit.

Dadurch lässt sich der durchschnittliche Schleppfehler bei Bewegungen reduzieren. Durch eine Kombination von Feed-Forward-Steuerung und PID braucht der PID-Regler nur noch den Restfehler zu korrigieren, der nach dem Vorsteuern verbleibt, was die Reaktion des Systems verbessert und eine sehr steife Regelung ermöglicht.

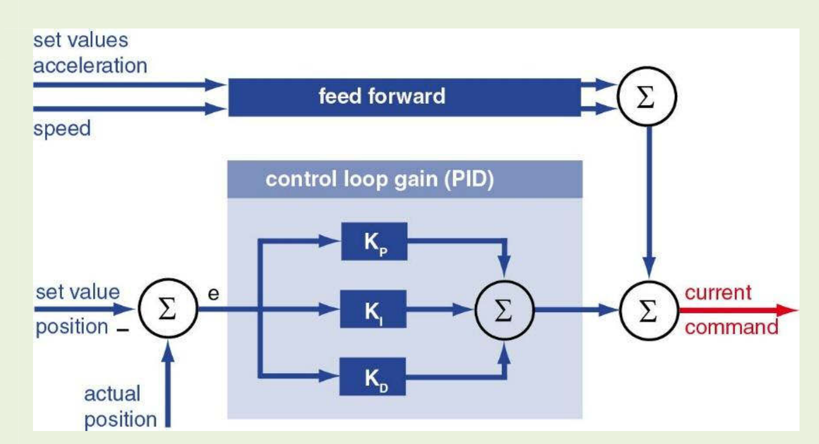


Abbildung 23: Schematische Darstellung der Vorsteuerung.
Die Feed-Forward-Werte werden aus der geforderten Beschleunigung und Drehzahl berechnet und zusätzlich zum PID Strom-Sollwert addiert.

Teil 2: Der Motion Controller

Im Teil 1 haben wir das System konfiguriert, den Motion Controller getunt und alle Einstellungen gespeichert. Der *EPOS2 P* Motion Controller ist nun zur weiteren Erkundung bereit.

- Kapitel 3 erforscht den Motion Controller, die *EPOS2 [internal]*, und benutzt die dazugehörigen Werkzeuge (Tools), um die verschiedenen Betriebsmodi zu studieren.
- Kapitel 4 untersucht die Ein- und Ausgänge, die zum Motion Controller gehören.
- Kapitel 5 ist optional. Es zeigt einige spezielle Betriebsmodi, die nicht auf die Online-Kommandierung zurückgreifen.

Bei uns übernimmt das *EPOS Studio* die Rolle des Masters. Vom Studio aus senden wir Einzelbefehle (und andere) und verfolgen direkt am Motor und im Studio wie sie ausgeführt werden. Im Moment brauchen wir nichts zu programmieren. Einfach spielen!



Master, **Slaves und on-line Kommandierung**
Die *EPOS2* Motion Controller sind Slaves im Netzwerk und erhalten ihre Befehle von einem übergeordneten System, *Master* genannt.

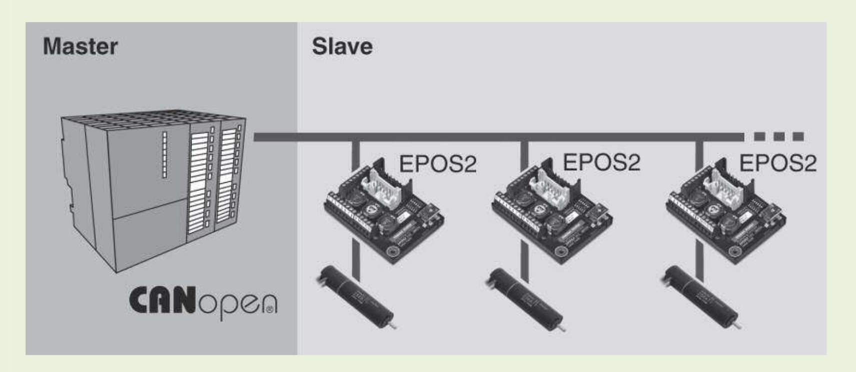


Abbildung 24: Master-Slave-Architektur mit EPOS2 *Slaves.*

Im Master läuft das Programm, das alle Prozesse und Slaves im Netzwerk kontrolliert. Als Master kommt eine SPS, ein Computer oder ein Mikroprozessor in Frage. Wie die Programmierung des Master (Prozesskontrolle) bei einer SPS gemacht werden kann, zeigt der Teil 3 dieses Buchs (Kapitel 7 ff).
In diesem Teil 2 dient aber das *EPOS Studio* als Master-Software.

Wichtig anzumerken ist, dass die *EPOS2* als Slave keine Befehlsfolgen speichern kann. Sie wird über Einzelbefehle on-line kommandiert, die vom Master gesendet und unmittelbar ausgeführt werden.

---

3 Den Motion Controller erkunden

Die Werkzeuge zur Erkundung des Motion Controller findest du im *Tools* Tab der *EPOS2 [internal]*.

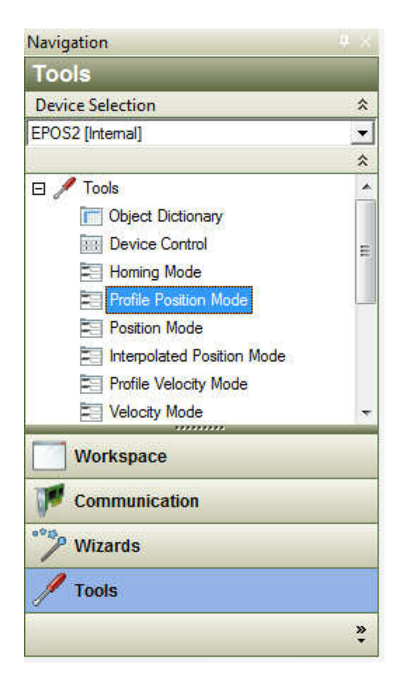
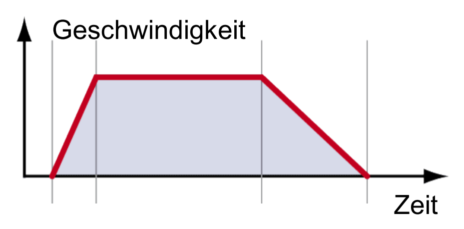


Abbildung 25: Wie man den Profile Position Mode *öffnet.*
Wähle den Tools Tab*, dann die EPOS2 [internal], und doppelklicke schliesslich auf* Profile Position Mode*.*

3.1 Profile Position Mode

Lernziele	Eine Punkt-zu-Punkt Bewegung ausführen. Die Bewegung mit dem eingebauten *Data Recorder* aufzeichnen.

Der *Profile Position Mode* ist der Standard-Positioniermodus für die meisten Anwendungen. Der Positioniervorgang folgt einem Geschwindigkeitsprofil, das automatisch im *Pfadgenerator* (Trajektorien-Generator) erzeugt wird. Der Pfadgenerator arbeitet mit einem Takt von 1 ms, angepasst auf die 1 kHz Abtastrate des Positionsregelkreises. Für jede Millisekunde wird ein neuer Positions-Sollwert (*Position Demand Value*) berechnet und in den Positionsregelkreis eingespeist. Der erzeugte Pfad berücksichtigt die Bewegungsparameter Beschleunigung, Geschwindigkeit, Zielposition, Profiltyp und andere.



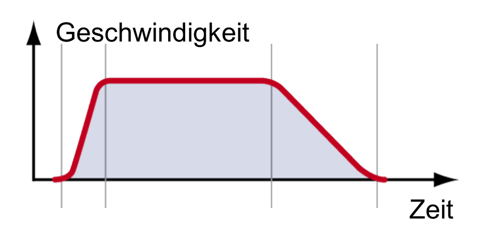


Abbildung 26: Geschwindigkeitsprofile für eine Positionierung. Links: Ein Trapezprofil (Geschwindigkeit gegen Zeit) mit schneller Beschleunigung und langsamerer Bremsrate. Rechts: Dasselbe Profil aber ruckarm mit gerundeten Ecken (Sinusprofil).

Punkt-zu-Punkt Bewegung

Hier ein erstes Schritt-um-Schritt Rezept, wie du eine Bewegung veranlassen kannst.

Schritt 1: Wähle das Werkzeug *Profile Position Mode* und aktiviere den Modus.
Schritt 2: Setze die Zielposition (*Target Position*) auf 20'000 qc.
Schritt 3: Setze die Profilgeschwindigkeit (*Profile Velocity*) auf 500 rpm.
Schritt 4: Ändere die Beschleunigung (*Profile Acceleration*) auf 10000 rpm/s.
Schritt 5: Ändere die Bremsrate (*Profile Deceleration*) auf 5000 rpm/s.
Schritt 6: *Enable* die *EPOS2*.
Schritt 7: Führe durch Klicken auf die Schaltfläche *Move Relative* eine Relativbewegung aus und beobachte wie der Motor dreht. Verfolge die Positionsänderung auch unten rechts im Dialogfenster.

Teste andere Bewegungen mit unterschiedlichen Parametern (*Profile* V*elocity*, *Target Position* …). Und sicher findest du den Unterschied zwischen *Move Absolute* und *Move Relative* heraus.



Grüne und rote LED : Anzeige des Zustandes der internen EPOS
Die grüne und rote LED der *EPOS2 P* zeigen den Zustand des internen *EPOS2* Motion Controller.

- Grüne LED blinkend — Endstufe gesperrt. Der Motor ist nicht bestromt.
- Grüne LED dauernd EIN — Endstufe enabled. Der Motor wird bestromt und geregelt.
- Rote LED EIN — Ein Fehler ist aufgetreten.

Den Zustand der LEDs kann man auch oben im Dialogfenster des EPOS Studio verfolgen.



Enable und Disable der Endstufe
Die Begriffe *Enabled* und *Disabled* beziehen sich auf den Zustand des Ausgangs der Motion Controller Endstufe.

- *Enabled* Die Endstufe ist mit dem Motor verbunden. Der Motor ist bestromt und kann geregelt werden.
- *Disabled* Der Ausgang der Endstufe ist vom Motor getrennt; es wird keine Drehmoment erzeugt. Der Regler wird aber nach wie vor mit Strom versorgt.

Bemerkung: Die Begriffe *Enabled* und *Disabled* werden ebenfalls benützt, um die Ausführung der digitalen und analogen Ein- und Ausgänge zu steuern (vgl. Kapitel 4).



Die Motion Control Regelkreise
In der Antriebstechnik ist die geregelte Grösse meist eine Kraft, ein Drehmoment, eine Geschwindigkeit oder eine Position. Entsprechend wird der Regler als Strom-, Drehzahl-, oder Positionsregler gestaltet. Oft kann man auch zwischen den verschiedenen Reglertypen oder Betriebsmodi wechseln.

Kraft/Drehmoment regeln
Dies ist die Kernfunktion des geregelten Antriebs, auf der auch die übergeordneten Drehzahl- und Positionier-Regelkreise aufbauen. Kräfte und Drehmomente werden benötigt, um Massen in Bewegung zu versetzen, zu bremsen und um die Reibkräfte zu überwinden. Oft finden sich auch Anteile der Gravitation, gegen die gearbeitet werden muss; sei es um Lasten zu heben, zu halten oder eine Abwärtsbewegung abzubremsen. In einigen Fällen der Drehmomentregelung wird aber auch nichts bewegt, sondern es soll nur mit einer definierten Kraft gegen ein Hindernis gedrückt werden.
Das im Motor erzeugte Drehmoment ist eine einfache Funktion des Motorstroms. Kräfte und Drehmomente kontrolliert zu erzeugen läuft also im Endeffekt darauf hinaus, den Motorstrom zu regeln.

Geschwindigkeit/Drehzahl regeln
Der Drehzahlregler versucht die gemessene Drehzahl mit der geforderten Drehzahl in Übereinstimmung zu bringen. Einen Körper mit einer bestimmten Geschwindigkeit zu bewegen oder zu rotieren heisst, den Körper zuerst auf die entsprechende Geschwindigkeit zu beschleunigen und anschliessend die angreifenden Kräfte zu kompensieren, welche die Geschwindigkeit beeinflussen. Dies geschieht dadurch, dass der Motor ein entsprechendes Drehmoment zur Drehzahländerung aufbringt. Dies läuft darauf hinaus, dass der Drehzahlregler dem untergeordneten Stromregelkreis einen entsprechenden Befehl erteilt (analog zum Positionsregler in Abbildung 27).

Position einnehmen und halten
Um einen Körper an eine bestimmte Endposition zu bringen, muss dieser zuerst in Bewegung versetzt (beschleunigt), verschoben und anschliessend abgebremst werden. Schliesslich gilt es die Endposition gegen allfällige Störkräfte zu halten. All dies verlangt nach Motordrehmoment, das gemäss den Bedürfnissen der übergeordneten Positionsregelung zur Verfügung gestellt werden muss. Im Endeffekt heisst auch dies wiederum Stromregelung.

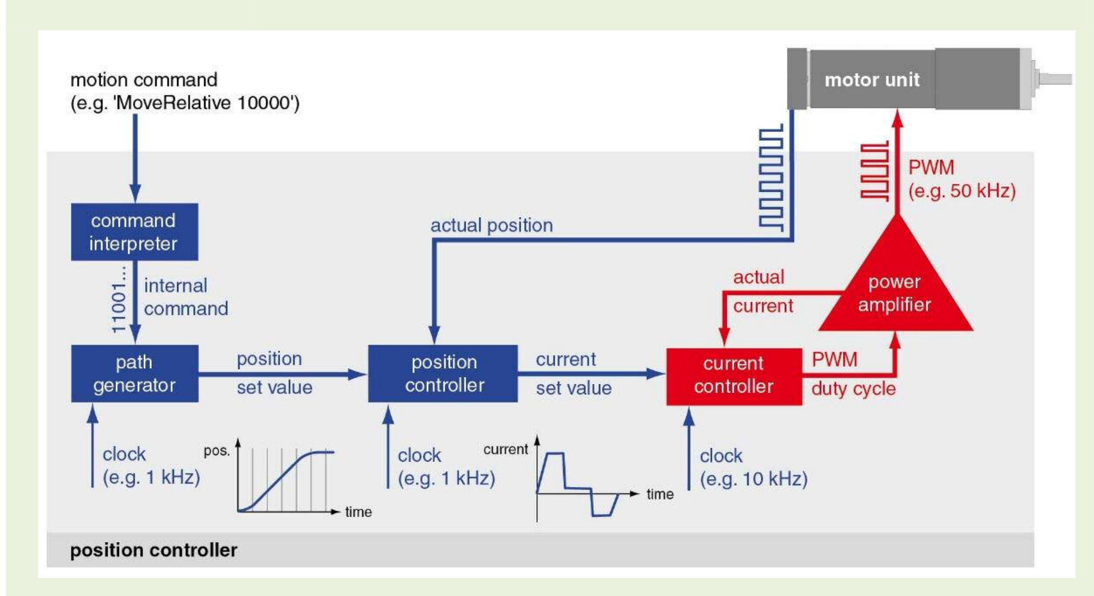


Abbildung 27: Darstellung einer Positionsreglung mit untergeordneter Stromregelung. Das Mastersystem sendet Bewegungsbefehle zum untergeordneten Motion Controller. Der Pfadgenerator verarbeitet die eingehenden Bewegungsbefehle und berechnet die Positionszwischenwerte für feste Zeitintervalle (1ms bei EPOS-Systemen) auf dem Weg zur Erreichung der Endposition. Diese Sollwerte werden dem Positionsregler periodisch zugeführt, der aus dem Vergleich mit der aktuellen Position die Sollwerte für den Stromregler bestimmt.
Positions- und Drehzahlregelkreis der EPOS2 *arbeiten mit einer Taktfrequenz von 1 kHz; der Regelkreis erhält jede Millisekunde eine neue Positionsinformation. Für die meisten Anwendungen ist dies schnell genug, da die mechanischen Reaktionszeiten mindestens einige Millisekunden betragen, wie man aus den mechanischen Zeitkonstanten der Motoren ersehen kann.*
Der Stromregler schliesslich steuert über die Endstufe (Leistungsverstärker) den Motorstrom, der zur mechanischen Reaktion des Antriebs führt. In der EPOS2 *hat der Stromregler eine Taktzeit von 0.1 ms oder 10 kHz. Dies ist 10-mal schneller als der Positionsregelkreis, was bedeutet, dass der Strom praktisch sofort da ist, wenn er vom Positions- oder Drehzahlregler benötigt wird.*

Daten-Recorder konfigurieren

Das *Data Recording* ist ein nützliches Werkzeug, um Bewegungen aufzuzeichnen und zu analysieren. Es funktioniert ganz ähnlich wie ein Speicheroszilloskop. Weitere Information findest du im entsprechenden Kapitel des Dokuments *Application Notes Collection*.

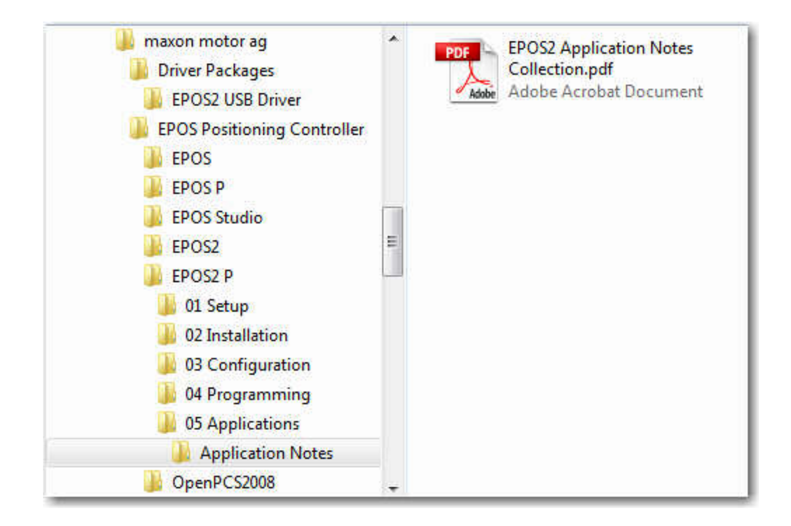


Abbildung 28: Wo man die Application Notes Collection *findet.*
Im Verzeichnis der maxon EPOS Software Installation.

Wir wollen den Recorder dazu verwenden, den Bewegungen besser folgen zu können. Hier findest du, wie er Schritt um Schritt konfiguriert wird.

- Schritt 1: Wähle das Tool *Data Recording*.
- Schritt 2: Klicke auf *Configure Recorder*.
- Schritt 3: Im Konfigurationsfenster wähle die Schaltfläche *Channel 1 Inactive*, um Kanal 1, sowie die Schaltfläche *Channel 2 Inactive*, um Kanal 2 zu aktivieren.
- Schritt 4: Wähle *Position Actual Value* (= aktueller Positionswert) vom Dropdown-Menü für Kanal 1 und *Position Demand Value* (= Positions-Sollwert) vom Dropdown-Menü für Kanal 2.
- Schritt 5: Ändere rechts oben die *Sampling Period* auf 2 ms.
- Schritt 6: Setze den Trigger Mode auf *Single Trigger Mode* und aktiviere die Checkbox *Movement Trigger*. Die Aufnahme wird somit bei der nächsten gestarteten Bewegung begonnen.
- Schritt 7: Klicke auf *OK*.

Die erste Aufnahme

Gehe zum Fenster *Profile Position Mode* und aktiviere den Modus. Führe eine Relativbewegung aus, z.B. mit den folgenden Parametern:

- *Target Position* 4000 qc.
- *Profile Velocity* 2500 rpm.
- *Profile Acceleration* 20000 rpm/s.
- *Profile Deceleration* 8000 rpm/s.

Überprüfe die Aufzeichnung im Fenster *Data Recording* und vergleiche die Sollposition mit der aktuellen Position (Istwert).



Data Recorder
Attached cursor: Mit dem *Attached Cursor* kannst du die Positionswerte überprüfen. Der Cursor wird an die erste dargestellte Kurve angeheftet. Benütze die Checkboxen um zu bestimmen, welche Kurven dargestellt werden sollen.

Zoom: Vergrössere das Diagramm, indem du den gewünschten Bereich mit der Maus markierst. Mit rechtem Mausklick kannst du wieder rauszoomen. Rein- und rauszoomen funktioniert auch schrittweise.

Speichere (*Save*) und exportiere (*Export*) die aufgezeichneten Daten in eine Textdatei (*.txt ASCII), die man beispielsweise im Microsoft Excel importieren kann! (Hinweis: Benutze die rechte Maustaste auf dem Diagramm). Öffne die exportierte Datei mit Notepad und/oder Excel. Andere Exportformate sind Bitmap oder das spezielle .rda Dateiformat.



Positioniergenauigkeit
In einem gut getunten System werden die Endpositionen bis auf einen Encoder-Quadcount (qc) genau erreicht. Sobald die Position um 1 qc neben der Zielposition liegt, wird der Positionsregler korrigierend eingreifen.
Bemerkungen: Die Systemreaktion auf Positionsabweichungen hängt von den Drehmoment- und Drehzahlmöglichkeiten des Systems, von der Reibung und den Massenträgheiten und von den Feedback- und Feed-Forward-Parametern ab. Die Regelparameter wurden dabei während des Tuningprozesses auf Grund des Systemverhaltens bestimmt. Man erkennt einmal mehr, wie wichtig es ist, das Tuning auf dem Gesamtsystem durchzuführen.

Ein weiterer Aspekt der Positioniergenauigkeit – neben diesem eher statischen Verhalten um die Endposition – betrifft wie die Zielposition erreicht wird. Das Verhalten kann mit mehreren Parametern umschrieben werden, wie in
Abbildung *29* gezeigt wird. Wiederum hängt das Resultat davon ab, wie das System getunt wurde und welchem Aspekt beim Tuning das meiste Gewicht beigemessen wurde: z.B. kein Überschiessen der Position erlaubt, oder sehr schnelle Positionierung (Überschiessen egal).

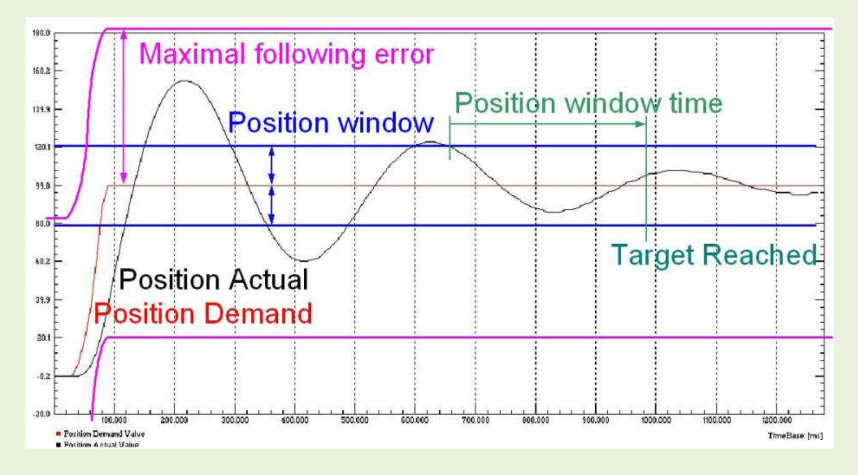


Abbildung 29: Target reached*.*
Die Parameter, die das Erreichen der Zielposition beschreiben (aus dem Dokument EPOS 2 Firmware Specification*). Die schwarze Kurve zeigt ein Überschiessen der Istposition (*Position Actual*). Die Zielposition wird als erreicht gewertet und angezeigt (*Target Reached*), sobald die Istposition länger als die spezifizierte Zeit (*Position window time*) innerhalb eines definierten Bereichs (*Position window*) um den Sollwert (*Demand value*) verbleibt.*

Limitierende Parameter bei Bewegungsprofilen

Einige allgemeine Parameter, die alle Bewegungsprofile einschränken, sind oben rechts im *Profile Position Mode* Fenster ersichtlich. Sie können definiert werden, um das System vor ungewollten und gefährlichen Bewegungen zu schützen.

Setze die *Max Profile Velocity* auf 1000 rpm und versuche eine Bewegung mit *Profile Velocity* von 1200 rpm zu starten.

Setze andere Grenzen und beobachte das Verhalten.



Maximaler Schleppfehler (*Maximum Following Error*)
Der *Max Following Error* ist die maximal zulässige Differenz zwischen Sollwert und Istwert der Position zu jedem Evaluationszeitpunkt. Er dient zur Sicherheit und zur Bewegungsüberwachung.
Wird der Schleppfehler zu gross, ist dies ein Zeichen, dass etwas schief läuft: Entweder kann der Antrieb die erforderliche Drehzahl nicht erreichen oder ist sogar blockiert. Allerdings sollte man ein gewisses Mass an Schleppfehler zulassen, da ja eine Feedback-Regelung nur mit einer Differenz zwischen Soll- und Istwert funktionieren kann. Verkleinere den *Max Following Error* somit nicht zu sehr.

3.2 Referenzfahrt (*Homing*)

Lernziele	Eine Referenzfahrt mit Stromschwelle durchführen. Die verschiedenen Parameter kennen, die im *Homing Mode* eingestellt werden können.

Da wir bis jetzt die Ein- und Ausgänge der *EPOS2* noch nicht behandelt haben, führen wir eine Referenzfahrt auf einen mechanischen Anschlag aus, ohne die Notwendigkeit Schalter anzuschliessen. (Das Homing mit Endschalter ist im Kapitel 4 beschrieben.) Folge einfach diesen Schritten:

Schritt 1: Wähle das *Tool Homing Mode* und aktiviere den Modus.
Schritt 2: Wähle die Homing Methode *Current Threshold Positive Speed.*
Schritt 3: Setze den Parameter *Current Threshold* auf 500 mA.
Schritt 4: *Enable* die *EPOS2*.
Schritt 5: Starte die Referenzfahrt mit der Schaltfläche *Start Homing*.
Schritt 6: Erzeuge eine Stromspitze, indem du kurz die Motorwelle mit der Hand blockierst. Für die *EPOS2* sieht das dann so aus, wie wenn der Motor in einen mechanischen Anschlag gestossen wäre und die Referenzposition wird detektiert!

Wiederhole diese Übung mit verschiedenen Parametern und versuche Antworten auf die folgenden Fragen zu finden:

- An welcher Stelle bleibt die Scheibe auf der Motorwelle stehen?
- Was ändert, wenn du die Homing Methode *Current Threshold Negative Speed* wählst?
- An welcher Stelle bleibt die Scheibe auf der Motorwelle stehen, wenn du die Homing Methode *Current Threshold Positive Speed & Index* wählst?
- Wenn du ein Homing mit Index (… *& Index*) ausführst: Welcher Unterschied besteht zwischen den Parametern *Speed Switch* und *Speed Index*?



Was bedeutet Referenzfahrt (*Homing*) und warum wird sie gebraucht?
Die Signale von Inkrementalencoder zeigen nur relative Positionsveränderungen (Positionsdifferenzen) an. Für eine absolute Positionierung muss das System zuerst auf eine Nullposition oder Referenzposition gefahren werden. Dieser Prozess heisst Referenzfahrt oder Homing. Alle weiteren Positionen des Mechanismus werden bezüglich dieser Referenzposition angegeben. Letztere wird meist über einen induktiven Näherungsschalter (z.B. als Referenzschalter oder Endschalter) oder mittels eines mechanischen Anschlags implementiert.

Homing mit Zusatzbewegung zum Indexkanal
Um die Wiederholgenauigkeit der Referenzposition zu erhöhen, kann eine zusätzliche Bewegung zur ersten Flanke des Indexkanal (Kanal I oder Z des Encoders) durchgeführt werden. Der Indexkanal I gibt einen schmalen Puls pro Umdrehung, was eine absolute Positionsbestimmung innerhalb einer Motorumdrehung ermöglicht. Bei dieser Homingmethode spielt es keine Rolle, wenn der Referenzschalter etwas früher oder später anspricht; die Referenzposition ist nun sehr genau auf die Zustandsänderung des Indexkanals gelegt.
In realen Systemen muss man aber aufpassen. Homing mit Indeximpuls kann nur funktionieren, falls der Indexpuls nicht gerade in den Fehlerbereich des Referenzschalters fällt. Darum vermeiden Maschinenbauer diese Funktionalität. Und wenn Motor oder Encoder ausgetauscht werden müssen, ist der Index nicht mehr an derselben Stelle. Das System muss neu kalibriert und/oder die neuen Positionswerte programmiert werden.

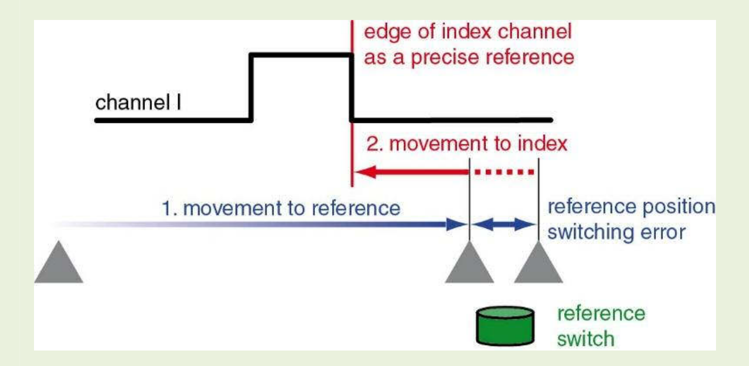


Abbildung 30: Prinzip des Homing *mit Indexkanal des Encoders. Der Bewegung zum Referenzschalter (oder mechanischen Anschlag) folgt eine Bewegung zurück zur Flanke des Indexkanals. Dies ergibt immer dieselbe Referenzposition, egal ob Positionsfehler beim Auslösen des Referenzschalters auftreten (z.B. aufgrund von Verschmutzung).*

3.3 Position Mode

Lernziel	Die Unterschiede zwischen dem *Profile Position Mode* und dem *Position Mode* erkennen.

Der *Position Mode* ermöglicht eine Positionierung ohne Profil, d.h. die Zielposition wird unmittelbar als neuer Sollwert für den Positionsregelkreis gesetzt. Der Pfadgenerator mit seiner Abfolge von Positionswerten kommt nicht ins Spiel. Im *Position Mode* ist nur absolute Positionierung möglich; es gibt keine Relativbewegungen.

Und wiederum zuerst ein einfaches Rezept um den *Position Mode* auszuprobieren.

Schritt 1: Setze die aktuelle Position auf 0, z.B. mit einem Homing. Dies vermeidet Probleme mit einem zu hohen Schleppfehler.

Schritt 2: Konfiguriere den *Data Recorder*:
Wähle als aufzuzeichnende Parameter *Position Demand Value* und *Position Actual Value*.
Setze den *Single Trigger Mode* und *Movement Trigger*.

Schritt 3: Wechsle zum Dialogfenster *Position Mode,* aktiviere den *Position Mode* und *Enable* den *EPOS2* Controller.

Schritt 4: Starte eine Bewegung zur Position 1000 qc (*Apply Setting Value*) und verfolge das Verhalten des Motors.

Schritt 5: Studiere das aufgezeichnete Diagramm.

Reduziere die maximale Drehzahl und Beschleunigung (*Max Profile Velocity, Max Acceleration*) im rechten Teil des Dialogfensters (*Parameters*). Beobachte wie das System gedämpfter reagiert.



Einschränkende Bewegungsparameter und Dämpfung
Du kannst das Systemverhalten im *Position Mode* dämpfen, indem du die *Max Profile Velocity* und die *Max Acceleration* oben rechts im *Position Mode* Dialogfenster reduzierst. Das bedeutet, dass diese beiden allgemeinen Grenzen auch beim Positionieren ohne Profil aktiv sind.
Grosse Positionierschritte können eine überaus starke Systemreaktion hervorrufen. Die benötigten Ströme können sehr hoch werden, da die *EPOS2* versucht, die Zielposition so schnell als möglich zu erreichen. Hohe Beschleunigungsströme können aber die Spannung im Netzgerät kurzzeitig zusammenfallen lassen - manchmal sogar unter die minimale Versorgungspannung der *EPOS2 P*, was einem Neustart gleichkommt.
Bemerkung: Solche allgemeinen Einschränkungen sind in allen Betriebsmodi aktiv. Sie können zu Fehlermeldungen führen, wenn das Anwendungsprogramm Befehle schickt, die im Widerspruch zu diesen Grenzen stehen; z.B. wenn ein Drehzahlprofil eine zu hohe Beschleunigung verlangt.



Positionieren ohne Profil
Der *Position Mode* ist ein nützlicher Betriebsmodus für sogenannte Slave-Achsen, die fortlaufend Positionssollwerte ohne grosse Sprünge erhalten. Der Pfadgenerator wird somit nicht benötigt. Ein Beispiel für einen fortlaufenden Positionssollwert ist eine analoge Sollwertspannung. Weitere Details in Kapitel 5.1.

Spezielle Betriebsmodi ohne Pfadgenerator sind *Master Encoder Mode*, wo die Slave-Achse den Signalpulsen eines externen Encoders (z.B. demjenigen der Master-Achse) folgt oder *Step Direction Mode*, wo jeder Befehlspuls einer Schrittmotor-Steuerung einem kleinen Drehwinkel entspricht. Weitere Details in den Kapiteln 5.2 und 5.3.

Schleppfehler (*Max following error*) und *Position Mode*
Setze den *Max Following Error* auf einen Wert, der höher ist als der grösste zu erwartende Positions-Schritt. Positionsschritte grösser als der *Max Following Error* sind nicht zulässig. Der Fehler wird ausgelöst, sobald die *EPOS2* versucht, den Befehl auszuführen, da der neue Sollwert (d.h. die Zielposition) zu weit von der aktuellen Position (d.h. der Startposition) weg ist.

3.4 Profile Velocity Mode

Lernziel	Eine feste Drehzahl in der *EPOS2* einstellen (Drehzahlregelung). Den Drehzahlverlauf im eingebauten *Data Recorder* verfolgen.

Der nächste Betriebsmodus, den wir untersuchen wollen, ist der *Profile Velocity Mode*. Dabei wird die Motordrehzahl und nicht die Position geregelt. Die neue Zielgeschwindigkeit wird mit einer Rampe angefahren - oder wenn du willst einem Geschwindigkeitsprofil, daher der Name des Modus - mit Beschleunigung- und Verzögerungswerten, die eingestellt werden können. Der *Profile Velocity Mode* ermöglicht somit graduelle Drehzahländerungen.

Als Ausgangspunkt für die Erforschung führe im *EPOS Studio* das folgende Bewegungsprofil aus und zeichne die Bewegung auf.

Schritt 1: Wähle das Dialogfenster *Data Recording* und konfiguriere den Recorder.
Aktiviere Kanal 1, 2 und 3.
Kanal 1: *Velocity Demand Value*, *Left Scale*
Kanal 2: *Velocity Actual Value*, *Left Scale*
Kanal 3: *Velocity Actual Value Averaged*, *Left Scale*
Wähle *Continuous Acquisition Mode*
Setze eine *Sampling Period* von 2 ms und drücke *OK*.

Schritt 2: Wähle unter *Tool* das *Profile Velocity Mode* und aktiviere den Modus.

Schritt 3: Setze eine*Target Velocity* von 2000 rpm. Wenn du eine relativ kleine Beschleunigung wählst (z.B. 200 rpm/s), kannst du sehr schön beobachten, wie die Scheibe mit der Motorwelle hochläuft.

Schritt 4: *Enable* die *EPOS2* und starte die Bewegung mit der Schaltfläche *Set Velocity*.

Schritt 5: Beobachte die Geschwindigkeitssignale auf dem *Data Recorder*.



Die Geschwindigkeitssignale im *Data Recorder* verstehen
Wenn du die Geschwindigkeitssignale im Data Recorder betrachtest, wirst du ein Bild wie in folgender Abbildung sehen.

Klar ist, die horizontale rote Linie ist der Geschwindigkeits-Sollwert von 2000 rpm.
Die schwarze Linie zeigt das Geschwindigkeitssignal des Sensors. Es schaut ziemlich verrauscht aus und eine genauere Betrachtung liefert, dass es in Schritten von 30 rpm rauf und runter springt. Weiter erkennt man eine gewisse Periode; das Signal wiederholt sich alle 30 ms.
Das periodische Verhalten kann man im grünen Signal, das die gemittelte Geschwindigkeit anzeigt, fast noch besser erkennen.

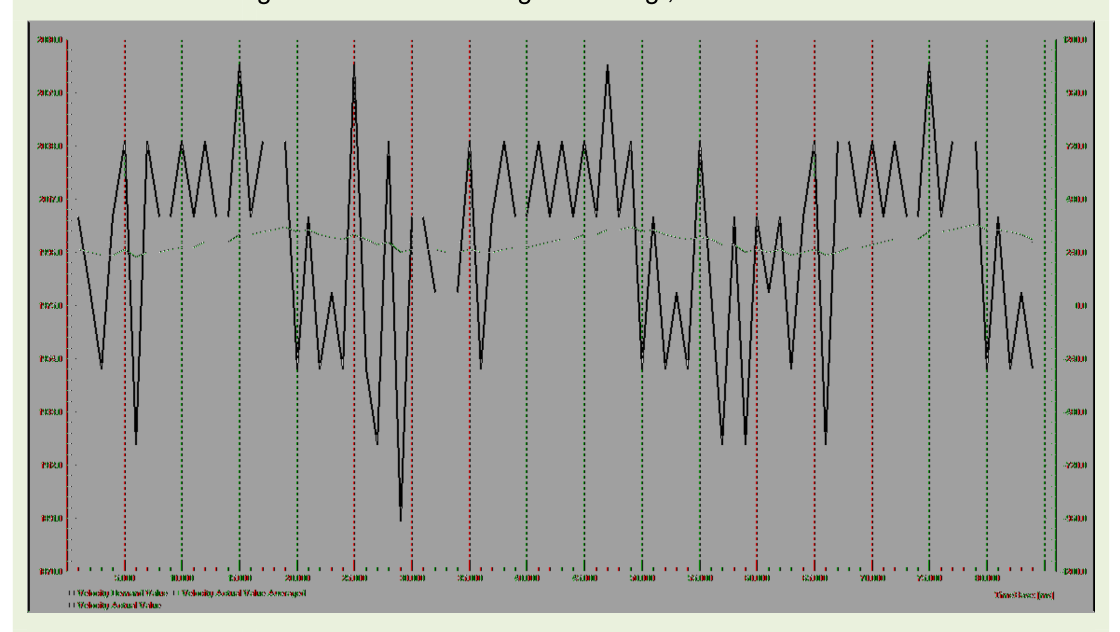


Abbildung 31: Geschwindigkeitssignale eines Motors mit MR Encoder*. Screenshot* Data Recorder*.*

Woher kommen die Schritte von 30 rpm im schwarzen Signal?
Sicher kann der Motor seine Drehzahl nicht so schnell ändern, wie das Signal vorzugeben scheint. Seine mechanische Zeitkonstante ist viel zu gross (> 4 ms). Was wir stattdessen beobachten, ist ein Quantisierungseffekt der Drehzahlmessung, die daher kommt, dass die Drehzahl aus der Positionsänderung pro Millisekunde berechnet wird. Die Position kann sich in Schritten von 1 quad count (qc) des Inkrementalencoders verändern. Eine Drehzahlvarianz von 1 qc pro Millisekunde ist gleichbedeutend mit 60'000 qc pro Minute und das entspricht 30 rpm bei einem Encoder mit 2000 qc pro Umdrehung.

Wenn du somit die wirkliche Drehzahl des Motors wissen willst, schau besser auf das grüne gemittelte Signal, den *Velocity Actual Value Averaged*. Dies macht mehr Sinn.
Übrigens, die Drehzahlgenauigkeit des gemittelten Signals entspricht etwa 3 rpm Abweichung bei einer Drehzahl von 2000 rpm. Und das ist schon mal nicht schlecht!

Woher kommt die Periodizität?
Die Periode im *Velocity Actual Value* und seinem gemittelten Wert korreliert mit der Umdrehung des Motors. Bei 2000 rpm benötigt der Motor genau 30 ms für eine Umdrehung, wie im Signal beobachtet.
Man kann sich zwei mögliche Ursachen für die periodischen Schwankungen vorstellen: Entweder kommen sie von einer Irregularität im Motor, z.B. eine erhöhte Reibung während einer Hälfte der Motorumdrehung, oder sie kommen von Ungenauigkeiten des Messsensors, d.h. des Encoders.
Hier ist fast sicher der zweite Grund der richtige. *MR Encoder* sind bekannt für diese Welligkeit. Ein Vergleich mit einem sehr genauen optischen Encoder auf demselben Motor zeigt deutlich den Unterschied (Abbildung 32). Optische Encoder sind bekanntermassen sehr präzise.

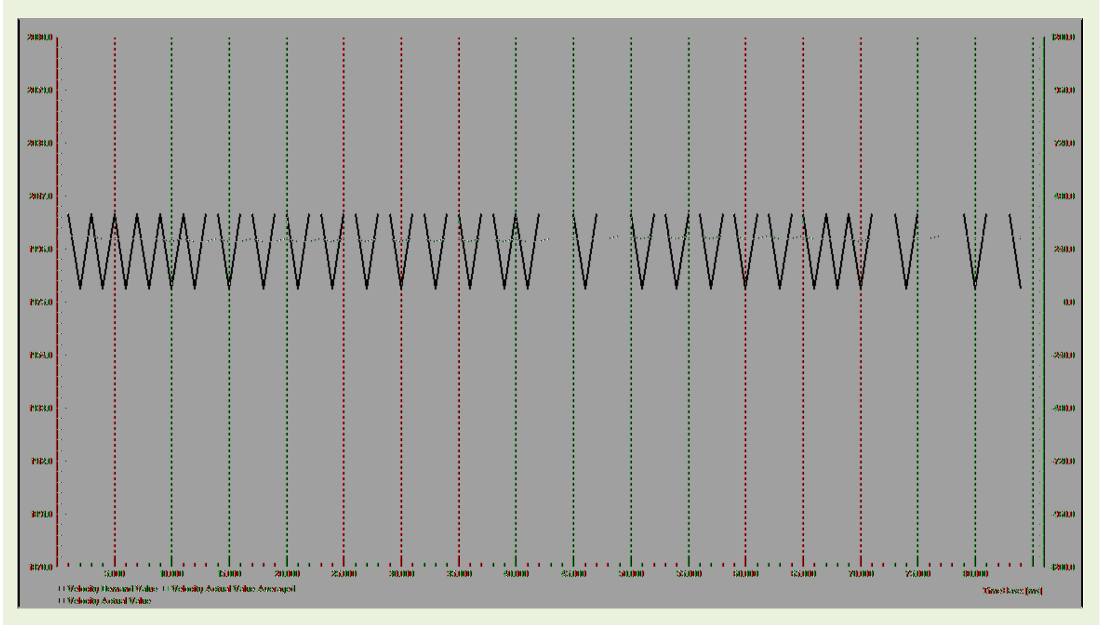


Abbildung 32: Geschwindigkeitssignale eines Motors mit optischem Encoder.
Screenshot Data Recorder.

3.5 Velocity Mode

Lernziel	Die Unterschiede zwischen dem *Profile Velocity Mode* und dem *Velocity Mode* erkennen.

Der *Velocity Mode* ist ganz ähnlich zum *Profile Velocity Mode*, allerdings ohne vordefinierte Drehzahlrampe. Noch immer sind die allgemein einschränkenden Bewegungsparameter aktiv. Beginne mit folgender Übung als Startpunkt für deine eigene Erforschung dieses Betriebsmodus.

Schritt 1: Wähle den Dialog *Data Recording* und konfiguriere den Recorder.
Aktiviere Kanal 1, 2 und 3.
Kanal 1: *Velocity Demand Value*, *Left Scale*
Kanal 2: *Velocity Actual Value Averaged*, *Left Scale*
Kanal 3: *Current Actual Value Averaged*, *Right Scale*
Wähle *Single Trigger Mode* und *Movement Trigger*
Setze eine *Sampling Period* von 2 ms und drücke *OK*.

Schritt 2: Gehe zum Dialogfenster *Velocity Mode* und aktiviere den Modus.

Schritt 3: Ändere die *Setting Value* auf 2000 rpm.

Schritt 4: *Enable* und starte die Bewegung mit der Schaltfläche *Apply Setting Value*.

Schritt 5: Beobachte die Drehzahl- und Stromsignale im *Data Recorder*. Beachte, wie der Strom ansteigt, sobald der Geschwindigkeitsfehler grösser wird.

3.6 Current Mode

Lernziel	Das Verhalten des Motors bei reiner Stromregelung kennen lernen.

Erforsche die Stromregelung, den *Current Mode*, mit beispielsweise folgenden Schritten.

Schritt 1: Wechsle zum *Tool Current Mode* und aktiviere den Modus.
Schritt 2: *Enable* das Gerät.
Schritt 3: Blockiere die Motorwelle (die Scheibe) von Hand! (Warum?)
Schritt 4: Setze den Stromsollwert (*Setting Value*) auf 100 mA und *Apply Setting Value.* Was beobachtest du, wie fühlt sich die blockierte Welle an?
Schritt 5: Erhöhe den Strom, z.B. auf 500 mA oder höher.



Ungewolltes Hochlaufen bei Stromregelung
Wenn du die Motorwelle bei Stromregelung nicht blockierst, läuft die Motordrehzahl unkontrolliert hoch. Warum?
Stromregelung bedeutet Drehmomentregelung. Der Regler versucht den Strom (das Drehmoment) auf dem eingestellten Wert zu halten:
Drehmoment am Motor heisst Beschleunigung
=> Höhere Drehzahl ergibt eine höhere induzierte Spannung (Gegen-EMK), die der anliegenden Spannung entgegenwirkt.
=> Eine höhere Spannung muss angelegt werden, um den Strom auf dem eingestellten Wert zu halten.
=> Höhere Spannung bedeutet, dass der Motor weiter beschleunigt.
=> höhere Gegen-EMK.
=> höhere Spannung muss angelegt werden.
=> ergibt höhere Drehzahl.
=> …
Als Resultat steigt die Motordrehzahl bis auf den Maximalwert an, der mit der verwendeten Spannungsversorgung möglich ist. Ein solches ungewolltes Hochlaufen wird durch einen übergeordneten Regelkreis (Geschwindigkeits- oder Positionsregler), durch einen mechanischen Anschlag oder eine zusätzliche Geschwindigkeitsüberwachung (wie die *Max Speed* Einstellung im Abschnitt *Current Mode* des Dialogfensters) verhindert.

Mit dem *Current Mode* haben wir die Erkundung der grundsätzlichen Motion Control Funktionalitäten abgeschlossen.

4 Das Werkzeug I/O Monitor

Lernziele	Benützen des *I/O Monitor* und praktisches Verstehen der I/O-Funktionalität und der Masken.

Auf der *EPOS2 [internal]* stehen 6 digitale und 2 analoge Eingänge, sowie 4 digitale Ausgänge zur Verfügung. Diese Ein- und Ausgänge sind primär dazu gedacht, spezielle Aufgaben im Umfeld der Antriebsachse zu übernehmen. Ein Beispiel ist der Endschalter, der anzeigt, wenn der mechanische Antrieb den sicheren Betriebsbereich einer Linearachse verlässt. Bewegt sich der Antrieb in den Endschalter, signalisiert der Motion Controller diesen spezifischen Fehler an den übergeordneten Master. Endschalter können aber auch für eine Referenzfahrt verwendet werden.

Die Ein- und Ausgänge der EPOS sind frei konfigurierbar. Du kannst sie für vordefinierte Funktionalitäten verwenden, oder für andere Zwecke (*General Purpose*). Einen der digitalen Eingänge kannst du dazu benützen, einen bestimmten Teil deines Programms der Master-SPS zu starten, oder das Signal eines Temperatursensors mit einem analogen Eingang lesen, oder das Ende einer Subroutine auf einem digitalen Ausgang anzeigen.

Beim *EPOS2 P Starter Kit* auf deinem Tisch werden auf der Leiterplatte die digitalen Eingänge über Schalter aktiviert. Die analogen Eingänge lesen die Spannung der Potentiometer und die digitalen Ausgänge schalten farbige LEDs ein. Öffne das *Tool I/O Monitor* und folge den Übungen, um das Verhalten und die Konfiguration der Ein- und Ausgänge kennen zu lernen. Eine detaillierte Beschreibung findest du in den einschlägigen Kapiteln der Dokumente *Application Notes Collection* und *Firmware Reference.*

4.1 Ausgänge

Als erste Übung setzen wir einen digitalen Ausgang und wir lernen, wie wir die physische Ausgabe maskieren können.

Schritt 1: Klicke auf die *Purpose* Spalte in der mittleren Tabelle des Dialogfensters und setze die digitalen Ausgänge (*Digital Output*) 1 bis 4 als *General A* bis *General D*. Setze die *Mask* auf *Enabled* und die Polarität (*Polarity*) auf *High Active*.

Schritt 2: Setze den *State* einiger digitaler Ausgänge auf *Active* und beobachte das Verhalten der LED-Ausgaben auf der Leiterplatte.

Schritt 3: Ändere die *Mask* auf *Disabled*. Welchen Einfluss hat dies auf das Setzen der Ausgänge und auf das Verhalten der LEDs?

Schritt 4: Ändere die *Polarity* auf *Low Active*. Wie beeinflusst dies die LED-Ausgabe?

Nun ordnen wir eine spezifische Funktionalität einem digitalen Ausgang zu.

Schritt 1: Ändere den *Purpose* vom *Digital Output 4* (oder einem anderen) auf *Ready*.

Schritt 2: Beobachte das Verhalten dieses Ausgangs während der nächsten Übungen, beispielsweise wenn ein Fehler auftritt.

4.2 Eingänge

Im Folgenden einige Übungen zu den Eingängen.
Zuerst lesen wir die analogen Eingänge: Drehe an den Potentiometern auf der Leiterplatte und beobachte die eingelesenen Werte im *I/O Monitor*.

Bemerkung: Es ist eine ziemliche Verzögerung zwischen der Aktivierung der Schalter und Potentiometer auf der Leiterplatte und der Anzeige auf dem *I/O Monitor*-Bildschirm zu beobachten. Auf Grund der recht grossen Datenmenge dauert die Aktualisierung der *I/O Monitor*-Bildschirmanzeige auf dem PC seine Zeit. Aber keine Angst, der interne Zustand der Eingänge wird sofort geändert.

Dann konfigurieren und lesen wir ein paar digitale Eingänge.

Schritt 1: Setze den *Purpose* der digitalen Eingänge (*Digital Input*) 1 bis 6 auf *General A* bis *General F*. Setze die *Mask* auf *Enabled* und die *Polarity* auf *High Active*.
Schritt 2: Schalte einen der digitalen Eingänge auf der Leiterplatte ein und beobachte die Reaktion auf dem *I/O Monitor*.
Schritt 3: Ändere die *Mask* auf *Disabled*. Wie beeinflusst dies das Lesen des Eingangs?
Schritt 4: Ändere die *Polarity* auf *Low Active*. Wie beeinflusst dies das Lesen des Eingangs?

Nun ordnen wir einem digitalen Eingang eine Funktionalität zu.

Schritt 1: Setze den *Purpose* von *Digital Input 1* auf *Negative Limit Switch* und *ExecMask* auf *Enable.*
Schritt 2: Schalte den *Digital Input 1* auf der Printplatte ein und beobachte das Verhalten der EPOS und des *Ready* Ausgangs.
Schritt 3: Lösche den Fehler im *EPOS Studio*.
Schritt 4: Was ist der Unterschied zwischen *Mask* und *ExecMask* (Ausführungsmaske)?

Und zum Schluss verwenden wir diese Funktionalität: Benütze den Digitaleingang 1, um eine Referenzfahrt (*Homing*) auf den *Negative Limit Switch* durchzuführen. Da du das Referenzieren in Kapitel 3.2 gelernt hast, solltest du fähig sein, dies ohne Anleitung auszuführen. Wie verhält sich der *Ready* Ausgang dabei?



Parameter speichern
Speichere die eingestellten Parameter der Ein- und Ausgänge mit einem Rechtsklick auf den Kopf des *I/O Monitor*-Dialogfensters und wähle *Save All Parameters*. Damit vermeidest du den Verlust deiner Einstellungen im Falle eines Stromunterbruchs.



Eingangsfunktionalitäten für Antriebsachsen

Endschalter (*Limit Switch*) und Referenzschalter (*Home Switch*)
Wir haben schon kennengelernt, wozu man die *Limit Switches* verwenden kann. Sie begrenzen beidseitig den mechanischen Arbeitsbereich. Oft werden sie mit induktiven Näherungsschaltern ausgeführt; nähert sich ein metallisches Teil, ändert sich der Zustand ihres Ausgangs. Wenn ein Endschalter anspricht, nimmt die Steuerung an, dass sich der Antrieb ausserhalb des zulässigen Bereichs befindet und erzeugt eine Fehlermeldung. Der Antrieb wird gestoppt und die Endstufe wird *Disabled*.
Man kann einen dieser Endschalter zur Referenzfahrt verwenden. Somit ist kein zusätzlicher Referenzschalter nötig und man weiss immer in welcher Richtung er zu finden ist. Benützt man einen Endschalter für die Referenzfahrt wird natürlich keine Fehlermeldung erzeugt.
Für die Referenzfahrt kann auch ein eigener Referenzschalter (*Home Switch*) mit zugehöriger Funktionalität verwendet werden.

Device Enable* und *Quick Stop
In den meisten Fällen wird die Endstufe des Controllers über Befehle vom Programm, das den Prozess im Master steuert, kontrolliert. Die *Enable* Eingangsfunktionalität ermöglicht das Ein- und Ausschalten der Endstufe über ein externes digitales Signal: Eine ansteigende Signalflanke bei *high active* Polarität an diesem Eingang wird die Endstufe einschalten (*Enable*) eine fallende Flanke wird sie sperren (*Disable*).
Diese Funktion kann in Situationen nützlich sein, wo man von aussen (d.h. ohne Programm) den Motor ausschalten will. Aber aufgepasst, diese Funktionalität entspricht nicht einem Not-Aus!
Der *Quick Stop* funktioniert ganz ähnlich, aber er stoppt die Achse und hält die Endposition. Der Motor ist noch immer bestromt, d.h. die Endstufe bleibt *Enabled*.

Position Marker
Der *Position Marker* Eingang merkt sich die momentane Position, an der dieser digitale Eingang aktiviert wird.
Hier ein typischer Anwendungsfall für diese Funktion: Wenn ein Objekt auf einem Förderband einen bestimmten Punkt passiert (z.B. eine Lichtschranke), so wird dies dem Motion Controller über den *Position Marker* Eingang signalisiert und die entsprechende Position des Förderbands wird gespeichert. Die Prozesssteuerung könnte nun berechnen, um wie viel das Band weiterlaufen muss, um das Objekt an einer definierten Stelle zum Stillstand zu bringen.

Ausgangsfunktionalitäten für Antriebsachsen

Ready/Fault
Der *Ready/Fault* Ausgang signalisiert die korrekte Funktionsweise des Geräts. Dies kennt man ja auch von den üblichen grünen und roten Leuchten bei Produktionsanlagen.

Position Compare
Position Compare ermöglicht es digitale Merkersignale an vordefinierten Positionen oder Positionsintervallen zu senden. Dies kann nützlich sein, um andere Geräte oder Prozesse über ein digitales Signal einzuschalten, wenn die Achse eine bestimmte Position erreicht hat.
Best practice: Wenn du den LED-Ausgang auf dem *EPOS2 P Starter Kit* blinken sehen willst, setze die *Pulse Width* auf den maximalen Wert, d.h. 64 000 μs.

Haltebremse (*Holding Brake*)
Mit Haltebremsen kann die Position gehalten werden, ohne dass der Motor bestromt werden muss. Damit kann man Energie sparen in Anwendungen, wo der Antrieb eine lange Zeit an einer festen Stelle bleiben muss und zum Halten der Position viel Drehmoment erforderlich ist (z.B. bei einer Vertikalachse). Haltebremsen halten die Position, wenn sie nicht bestromt sind und können deshalb auch zur Notbremsung bei Stromausfall verwendet werden.
Der leistungsstarke *Holding Brake* Ausgang ermöglicht es die Bremse direkt und ohne separates Netzteil anzusteuern unter Berücksichtigung der Aktivierungszeit der Bremse.

5 Alternative Betriebsmodi (optional)

Die alternativen Betriebsmodi ermöglichen einen Betrieb des Motion Controller ohne on-line Kommandierung. An ihrer Stelle werden analoge Spannungssignale oder digitale Signalpulse als Befehle verwendet. Die einkommenden Signale werden zu Sollwerten konvertiert und direkt in den entsprechenden Regelkreis gespeist, ohne Bewegungsprofil oder Drehzahlrampe.

Beachte aber, dass diese Betriebsmodi nicht der ursprünglichen Idee der EPOS Produktfamilie entsprechen. Die EPOS wurde als System entwickelt, das die Befehle on-line über einen CAN-Bus von einem Masterprogramm erhält.

5.1 Analoger Sollwert

Beim Betrieb mit analogem Sollwert wird die Sollwertvorgabe für die Position, die Geschwindigkeit oder den Strom an einem der analogen Eingänge eingelesen, umgerechnet und dem entsprechenden Regelkreis zugeführt. Testen wir dies mit Positions- und Drehzahlregelung aus.



Hardware Enable
Da diese alternativen Betriebsmodi nicht on-line kommandiert sind, sollten wir eine Möglichkeit vorsehen, die Endstufe unabhängig ein- und auszuschalten. Dazu konfigurieren wir einen der digitalen Eingänge als *Enable.* Verwende dazu den *I/O Monitor* und wähle bei einem Eingang *Drive Enable* als *Purpose*. Vergiss nicht, die Masken richtig zu setzen.

Analoge Drehzahlregelung

Lernziele	Einrichten der analogen Drehzahlregelung. Betrieb ohne serielle on-line Kommandierung.

Zuerst sollten wir der *EPOS2 [internal]* sagen, dass sie die Sollwerte über einen analogen Eingang erhält. Benütze den *I/O Monitor*, um einen der analogen Eingänge als *Velocity Setpoint* zu konfigurieren.

Schritt 1: Wechsle zum Dialogfenster *I/O Monitor*.
Schritt 2: Ändere den *Purpose* des *Analog Input 1* auf *Velocity Setpoint* (ist nur möglich wenn der Antrieb *Disabled* ist).
Schritt 3: Klicke auf die *Show Attributes* Schaltfläche unten.
Schritt 4: Gib einen Skalierungsfaktor (*Setpoint Scaling*) ein: das Verhältnis zwischen analoger Sollwertspannung und Drehzahl (z.B. 1000 rpm/V).
Schritt 5: Optional: Speichere die Parameter (z.B. im Kontextmenu: rechter Mausklick).

Nun führen wir die Drehzahlregelung mit analogem Sollwert aus.

Schritt 1: Wechsle zum Dialogfenster *Velocity Mode* und aktiviere den Modus.

Schritt 2: Drehe das Potentiometer auf das Minimum. Beobachte die Veränderungen der Angaben unten rechts im Dialogfenster.

Schritt 3: Aktiviere das Einlesen des analogen Sollwerts. *Enable* die Checkbox *Execution Mask*. (Nach diesen Konfigurationsschritten könntest du das USB-Verbindungskabel ausstecken. Wenn du jedoch die Sollwerte, die Signale des Potentiometers und die Istwerte auf dem Bildschirm verfolgen willst, lass das USB-Kabel eingesteckt.)

Schritt 4: *Enable* die *EPOS2*, stelle verschiedene Drehzahlwerte mit dem Potentiometer ein und beobachte das Motorverhalten.

Schritt 5: Versuche auch verschiedene Einstellungen des *Setpoint Scaling* und *Setpoint Offset*.



Limitierung und Dämpfung der Systemreaktion
Du kannst das Systemverhalten dämpfen, indem du die Parameter *Max Profile Velocity* und *Max Acceleration* oben rechts im Dialogfenster reduzierst. Dies ist vor allem bei verrauschtem Analogsollwertsignal zu empfehlen.

Analoge Positionierung

Lernziele	Einrichten der analogen Positionsregelung. Betrieb ohne serielle on-line Kommandierung.

Benütze den *I/O Monitor*, um einen der analogen Eingänge als *Position Setpoint* und einen der digitalen Eingänge als *Drive Enable* zu konfigurieren (geht nur falls der Antrieb *disabled* ist).

Schritt 1: Führe eine Referenzfahrt (*Homing*) aus: Setze die *Home Position* auf 0.

Schritt 2: Wechsle zum Dialogfenster *Position Mode* und aktiviere den Modus.

Schritt 2: Drehe das Potentiometer auf das Minimum. Beobachte die Veränderungen der Angaben unten rechts im Dialogfenster.

Schritt 4: Gib einen Skalierungsfaktor (*Setpoint Scaling*) ein: das Verhältnis zwischen analoger Sollwertspannung und Position (z.B. 2000 qc/V).

Schritt 5: Aktiviere das Einlesen des analogen Sollwerts. *Enable* die Checkbox *Execution Mask*. (Nach diesen Konfigurationsschritten könntest du das USB-Verbindungskabel ausstecken. Wenn du jedoch die Sollwerte, die Signale des Potentiometers und die Istwerte auf dem Bildschirm verfolgen willst, lass das USB-Kabel eingesteckt.)

Schritt 6: *Enable* die *EPOS2*, erhöhe die Position durch Drehen des Potentiometers und beobachte das Motorverhalten.

Schritt 7: Versuche auch verschiedene Einstellungen des *Setpoint Scaling* und *Setpoint Offset*.

5.2 Master Encoder Mode: Elektronisches Getriebe

Im *Master Encoder Mode* werden die Signalpulse eines externen Inkementalencoders (Kanal A und B) als Positionssollwerte interpretiert. Ist dieser Modus eingestellt, versucht die *EPOS2 [internal]* der Bewegung des externen Encodersignals zu folgen.

Zwischen den Impulsrate des externen und des internen Encoders kann ein festes Verhältnis festgelegt werden sowie die relative Drehrichtung. Für die Geschwindigkeit ist das Resultat ähnlich wie zwischen Ein- und Ausgang bei einem Getriebe; daher der Name *elektronisches Getriebe*.

Leider ist zu diesem Thema mit der Hardware, die wir auf dem Tisch haben, keine praktische Arbeit möglich. Wir haben keinen externen Encoder.

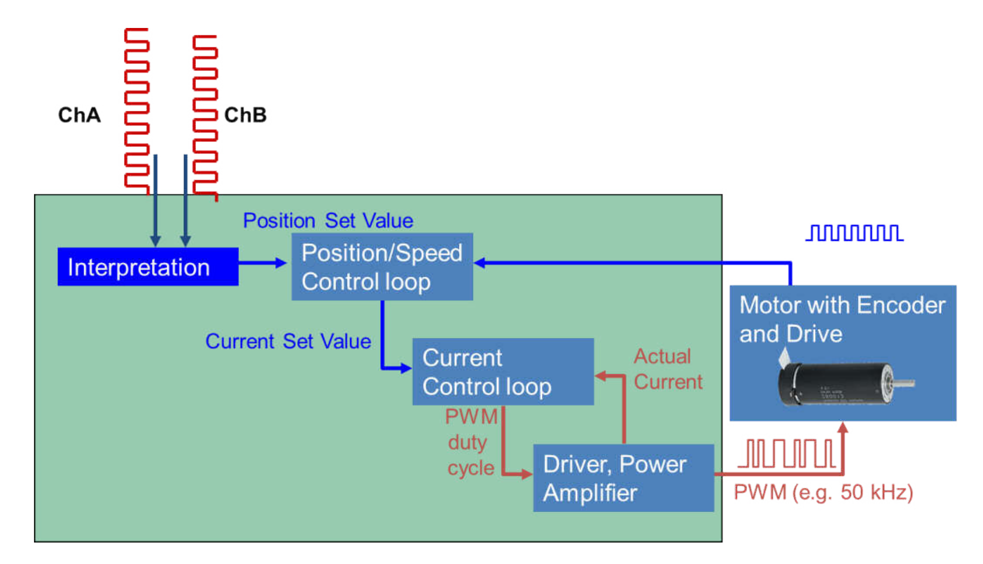


Abbildung 33: Schematische Darstellung des Master Encoder Mode.

5.3 Step Direction Mode

Im *Step Direction Mode* werden einkommende Signalpulse als Positionierschritte (*Step*) interpretiert. Der Zustand eines zweiten Signals ergibt die Drehrichtung (*Direction*). Die *EPOS2 [internal]* folgt Schritten am Signaleingang. Dabei kann man definieren, wie viele Encoder-Quadcounts einem Schrittimpuls entsprechen.

Schritt- und Drehrichtungssignale können von vielen SPS generiert werden, meist um Schrittmotoren anzusteuern. Mit einem EPOS2 Motion Controller und einem DC Motor könnte man einen Schrittmotor samt Steuerung ersetzen, z.B. in Anwendungen, wo sich der Schrittmotor als zu schwach oder zu gross erweist.

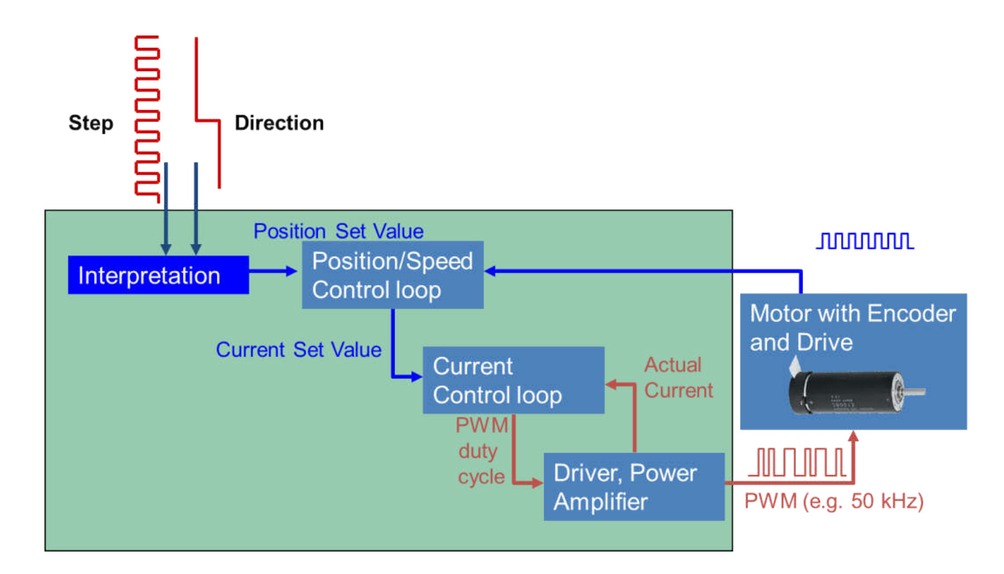


Abbildung 34: Schematsche Darstellung des Step Direction Mode.

Simulation des Step Direction Mode *mit I/Os*

Lernziele	Den *Step Direction Mode* einrichten und die digitalen Eingänge zur Simulation für Schritt- und Drehrichtungssignale verwenden. Manueller Betrieb ohne serielle on-line Kommandierung.

Schritt 1: Benutze den *I/O Monitor* um die digitalen Eingänge 2 und 3 als *General Purpose* A und B zu konfigurieren. Speichere die Parameter.

Schritt 2: Konfiguriere einen der restlichen digitalen Eingänge als *Drive Enable*. Setze *Mask* und *Exec Mask* auf *Enabled*.

Schritt 3: Öffne das Tool *Step Direction Mode* und aktiviere den Modus.

Schritt 4: Gib einen *Scaling factor*, d.h. um wie viele Quadcounts soll sich der Motor pro Signalpuls auf dem digitalen Eingang 3 weiterbewegen. Z. B. setze 200 qc pro Puls, dies entspricht bei uns einer zehntel Motorumdrehung.

Schritt 6: Reduziere die *Max Acceleration* (z.B. < 100 000 rpm/s), um die Systemantwort zu dämpfen und um zu vermeiden, dass die Stromversorgung aufgrund des hohen benötigten Stroms zeitweise in die Knie geht.

Schritt 5: *Enable* die Achse mit dem entsprechenden digitalen Eingangsschalter und beobachte, wie sich der Motor je nach Stellung des digitalen Eingangs 2 (*Direction*) verhält, wenn der digitale Eingang 3 (*Step*) betätigt wird. Verfolge die Position im Dialogfenster. Teste verschiedene Einstellungen der Parameter im Dialogfenster.

Weitere Angaben finden sich in Kapitel 5 des Dokuments *EPOS2 Application Notes Collection*.

Intermezzo: CAN und CANopen

Die *EPOS2 P* ist ein *CANopen Gerät*, oder genauer, es sind zwei CANopen Geräte, ein IEC-61131-programmierbares Gerät und ein Motion Controller. Der Zweck dieses Kapitels ist, dich mit den wichtigsten Eigenschaften und den Ideen hinter CANopen-Geräten (*CANopen Devices*) vertraut zu machen. Damit erhältst du ein besseres Verständnis, wie die Kommunikation funktioniert und dies ist für die spätere Programmierung nützlich.

Allerdings ist dies nur eine Einführung und nicht eine volle Beschreibung von CANopen. Auf der CiA (CAN in Automation)-Website findest du weitere Informationen (www.can-cia.org).

6 Eine Einführung in CANopen und CAN

CANopen ist ein Kommunikations-Protokoll, das üblicherweise auf dem CAN-Feldbus aufsetzt. CAN deckt die tieferen Kommunikationsschichten ab und hat den Schwerpunkt auf technischen Aspekten, während CANopen die Bedeutung der Daten und die Geräte im Netzwerk spezifiziert. Anders und etwas vereinfacht gesagt: Der CAN-Bus beschreibt das Vehikel, wie Daten übertragen werden, und das CANopen-Protokoll sagt, was für Daten übertragen werden.
CANopen ist ein Standard, der von der unabhängigen Anwenderorganisation *CAN in Automation* unterhalten und überwacht wird.

Wir wollen uns hier nicht damit auseinander setzen, wie genau Meldungen übertragen und empfangen werden. Alles was wir wissen müssen, ist, dass CANopen und CAN einen zuverlässigen und kostengünstigen Rahmen zur Verfügungstellen, um Informationen zwischen Geräten auszutauschen.
Einige Einschränkungen:

- Maximale Anzahl Knoten pro Bus — 127 Knoten
- Maximale Bitrate — 1 Mbit/s bis 40 m totale Buslänge
Alle Knoten müssen dieselbe Bitrate haben!
- Typische CAN-Telegrammlänge — 100 – 130 Mikrosekunden bei maximaler Bitrate

Die wichtigsten Konzepte, mit denen wir uns auseinander setzen müssen, sind das Gerätemodell (*Device Model*) und das darin enthaltene Objektverzeichnis (*Object Dictionary*). Zuerst aber ein paar Worte zu CAN und CANopen.

6.1 CAN

CAN steht für *Controller Area Network*. Dieser Feldbus wurde ursprünglich für Anwendungen im Automobilbau entwickelt, wird nun aber auch in anderen Bereichen wie der Industrieautomation und in Medizinalgeräten eingesetzt.

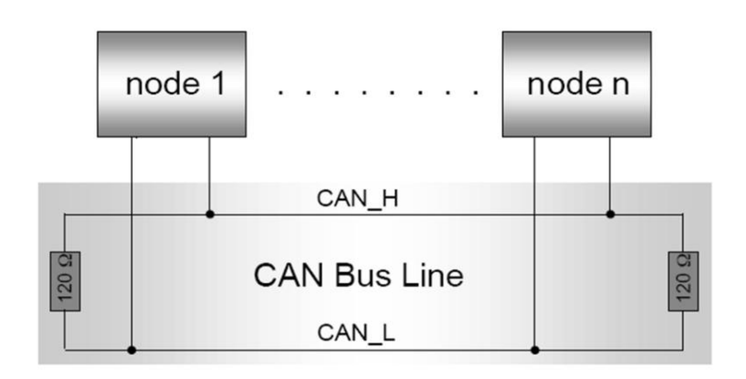


Abbildung 35: Physikalischer Aufbau des CANbus.

Physikalisch betrachtet, wird jedes Gerät – Knoten (*node*) genannt und mit individueller Adresse (*node number*) – parallel an die Busleitung angeschlossen. Der Bus überträgt die Signale differentiell, was einen korrekten beidseitigen Abschluss der Leitung verlangt (120 Ohm Widerstand), um Signalreflektionen zu vermeiden. Die Signalbits benötigen eine gewisse Zeit, um sich über den ganzen Bus auszubreiten. Darum hängt die Übertragungsrate von der Buslänge ab. Maximal kann eine Bitrate von 1 Mbit/s bei Buslängen bis etwa 40 m erreicht werden.

"Der *Transfer Layer* repräsentiert den Kern des CAN-Protokolls." Dort werden Struktur, Übertragung und Empfang der Nachrichten definiert. "Der Transfer Layer ist verantwortlich für das Bit-Timing und die Synchronisation, das Verpacken von Nachrichten, die Arbitrierung und Bestätigung, die Fehlerentdeckung, -anzeige und -eingrenzung." (übersetzt aus dem englischen Wikipedia)

Kurz gesagt, bietet CAN einen Rahmen, in dem Mikrokontroller und andere Geräte sicher und zuverlässig miteinander kommunizieren können.

6.2 CANopen

"In den Begriffen des OSI-Modells implementiert CANopen die höher liegenden Kommunikationsschichten; *Network layer* und höher. Der CANopen-Standard besteht aus einem Addressierungsschema, einigen kleinen Kommunikationsprotokollen und einem Anwendungslayer, das durch Geräteprofile festgelegt ist. Die Kommunikations-protokolle unterstützen das Netzwerkmanagement, die Geräteüberwachung und die Kommunikation zwischen den Knoten, einschliesslich einer einfachen Transportschicht für die Segmentierung und Desegmentierung von Nachrichten. Die unteren Protokoll-Level (Data-Link und physikalische Schichten) werden meist als Controller Area Network (CAN) implementiert, obwohl auch andere Kommunikationarten (wie Ethernet Powerlink, EtherCAT) das CANopen Geräteprofil einsetzen können." (übersetzt aus dem englischen Wikipedia)

Kurz gesagt, benützt CANopen den CAN-Rahmen, um sinnvolle Meldungen zwischen definierten Knoten (*Devices*) in einem Netzwerk zu senden.

.

6.3 CANopen Geräteprofil

Das CANopen Geräteprofil ist ein zentrales Element von CANopen. Es beschreibt, wie die Knoten in einem Netzwerk strukturiert sein müssen. Man kann sich ein CANopen Gerät aus drei Untereinheiten bestehend vorstellen: einer *Kommunikationseinheit*, dem Objektverzeichnis (*Object dictionary)* und dem *Anwendungsteil* (Abbildung 36, von unten nach oben). Zusätzlich braucht es eine Zustandsmaschine, um das Gerät zu starten und zurückzusetzen. Sie muss die Zustände *Initialisierung, Pre-operational, Operational* und *Angehalten* enthalten. Typischerweise wird der *Pre-operational* Zustand zur Konfiguration benützt, während Echtzeitkommunikation auf den Zustand *Operational* beschränkt ist.

Die *Kommunikationseinheit* ist für die Kommunikation mit den anderen Knoten im Netzwerk zuständig. Sie enthält alle CAN und CANopen Kommunikationsmerkmale: Netzwerk-Management-Informationen empfangen und senden, als auch Daten ins Objektverzeichnis schreiben oder von ihm lesen.

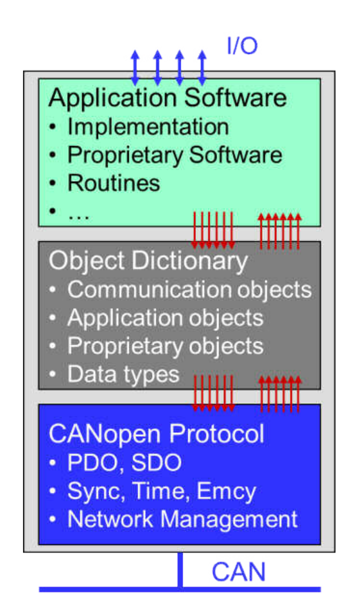


Abbildung 36: CANopen Geräteprofil

Der *Anwendungsteil* führt die spezifischen Funktionen des Gerätes aus. Dies kann ein Motion Controller wie die *EPOS2 [internal]*, eine programmierbare Steuerung wie die *EPOS2 P* oder sonst eine Funktionalität sein. Die Anwendung wird über Parameter und Variablen im *Objektverzeichnis* konfiguriert und gesteuert.

Objektverzeichnis (Object dictionary)

Das *Objektverzeichnis* ist das Herz des Geräts. Es verbindet die Welt der CANopen-Kommunikation und die Anwendung. Die Anwendung benützt die Daten des Objektverzeichnisses als Eingangsgrössen, um ihre Aufgabe auszuführen, und sie schreibt die Resultate und Ausgangsgrössen in dasselbe Objektverzeichnis. Andererseits beruht die ganze Kommunikation zwischen Geräten im CANopen-Netzwerk auf dem Datenaustausch zwischen den entsprechenden Ojektverzeichnissen. Somit läuft Kommunikation mit einem CANopen-Gerät im Endeffekt darauf hinaus, Parameter zwischen entsprechenden Objektverzeichnissen auszutauschen.

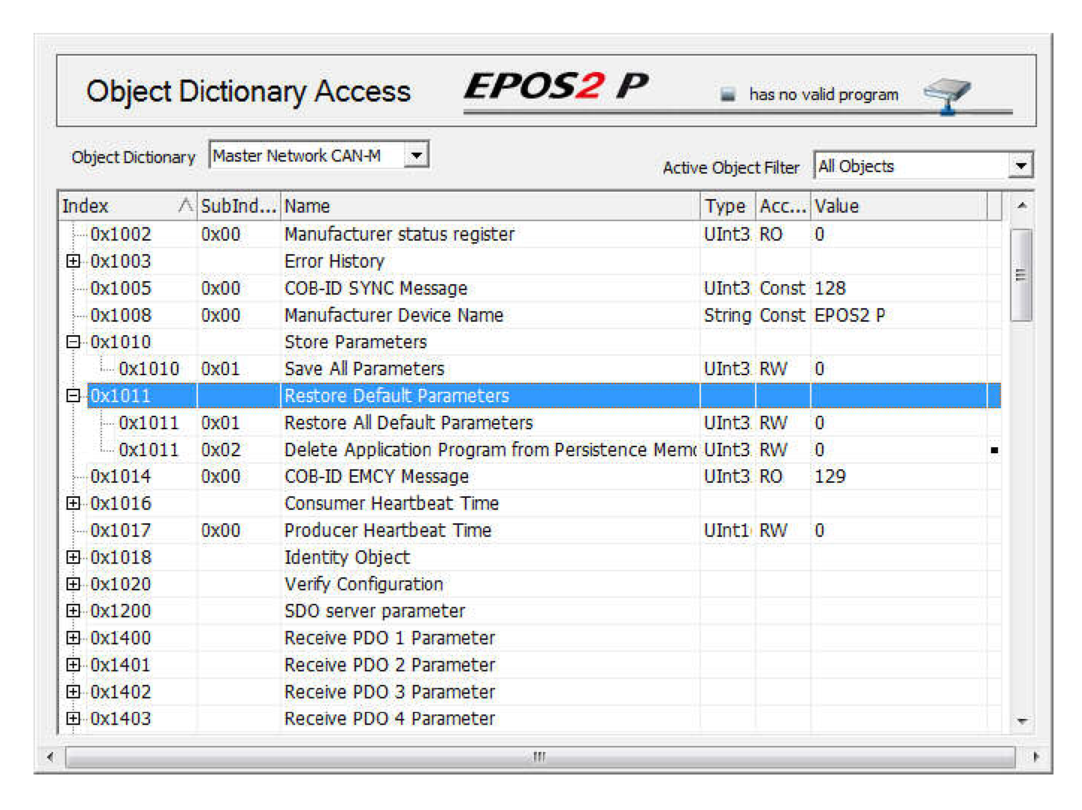


Abbildung 37: Zugriff auf das Objektverzeichnis (Object Dictionary) *der* EPOS2 P *im* EPOS Studio Tools *Dialogfenster.*

Das Objektverzeichnis besteht im Wesentlichen aus einer Tabelle von Parametern, die alle Eigenschaften des Geräts seine Kommunikationskanäle, seine Konfiguration und Einstellungen beschreiben, aber auch die Konfiguration der Anwendung und ihre Ein- und Ausgabedaten.

Ein Eintrag im Objektverzeichnis ist bestimmt durch

- Adresse — (16-bit Index und 8-bit Subindex)
- Name — ein String, der den Eintrag beschreibt
- Typ (*Type*) — der Datentyp des Parameters
- Zugriff (*Access*) — lesen und schreiben (*RW*), nur lesen (*RO*), nur schreiben (*WO*) oder konstant (*const*)
- Wert (*Value*) — des Parameters

Die grundlegenden Datentypen für Werte im Objektverzeichnis wie Booleans, Integers und Gleitkommazahlen sind im CANopen-Standard definiert. Daneben sind aber auch zusammengesetzte Datentypen wie Arrays, Records und Strings möglich.



Objektverzeichnisse der *EPOS2 P*
Die komplexe Struktur der *EPOS2 P* findet man in den verschiedenen Objektverzeichnissen und CAN-Ports wieder. Die eingebaute SPS hat eine *CAN-I* Verbindung zum Slave *EPOS2 [internal]*, der *CAN-S* Port kann für weitere Slave-Geräte verwendet werden, und der *CAN-M* Port für Befehle einer noch höheren Kontrollebene. Jede dieser drei CAN Schnittstellen hat ihr eigenes Objektverzeichnis, ebenso wie die *EPOS2 [internal]*.

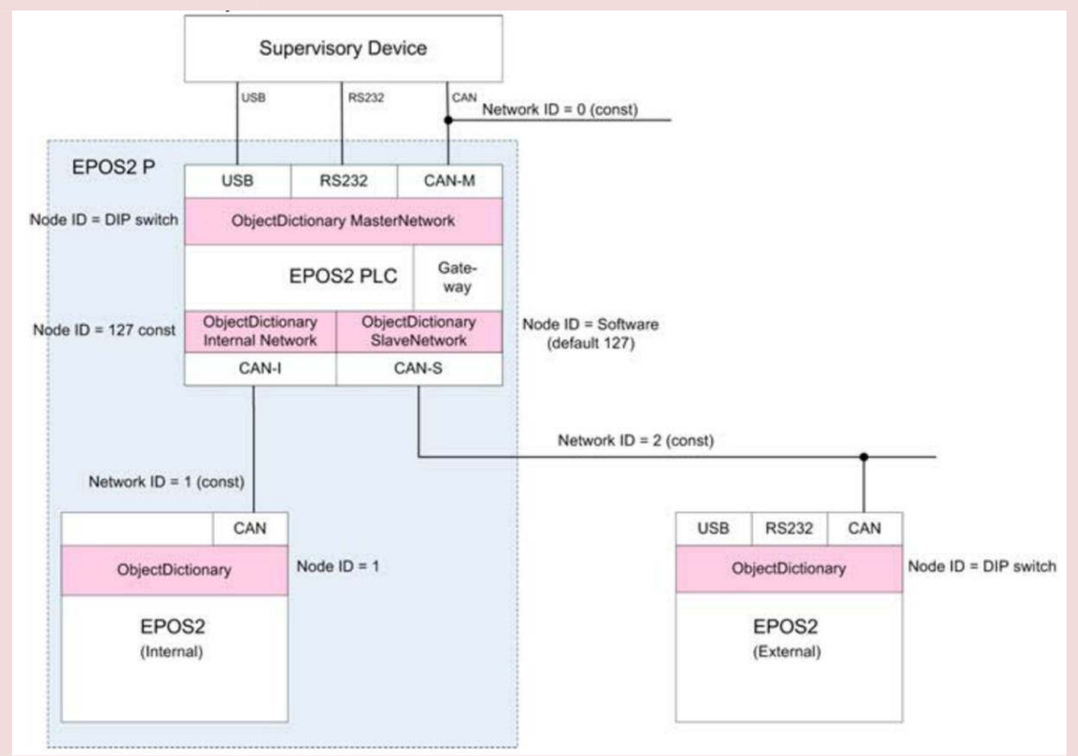


Abbildung 38: Die verschiedenen Object Dictionaries *im* EPOS2 P *Netzwerk.*

Einschränkungen der Variablentypen bei EPOS-Systemen
EPOS2-Systeme beinhalten keine Gleitkommazahlen (*floating numbers*). Numerische Variablen können nur ganzzahlig (*integer*) von unterschiedlicher Grösse sein.

Firmware Specification
Alle Objekte der *EPOS2 P* und der *EPOS2 [internal]* sind in der entsprechenden *Firmware Specification* beschrieben. Darin findest du auch Angaben zu den Wertebereichen und zur Bedeutung der Parameterwerte (Einheiten).

Standard CANopen Geräteprofile

Objektverzeichnisse können nicht beliebig festgelegt werden. Zuerst einmal gilt es Regeln zu beachten, welche Daten wo im Objektverzeichnis abgelegt werden müssen. Weiter existieren Standard-Geräteprofile für eine Anzahl Anwendungen. Die wichtigsten Profile für uns sind die Nummern 402 und 405 für Motion Controller und programmierbare Geräte.

Profile number	Device class
CiA 401	Generic I/O Modules
CiA 402	Drives and Motion Control
CiA 404	Measuring devices and Closed Loop Controllers
CiA 405	IEC 61131-3 Programmable Devices
CiA 406	Rotating and Linear Encoders
CiA 408	Hydraulic Drives and Proportional Valves
CiA 410	Inclinometers
CiA 412	Medical Devices
CiA 413	Truck Gateways
CiA 414	Yarn Feeding Units (Weaving Machines)
CiA 415	Road Construction Machinery
CiA 416	Building Door Control
CiA 417	Lift Control Systems
CiA 418	Battery Modules

Abbildung 39: Einige Standard CANopen Geräteprofile (Quelle CiA Website).

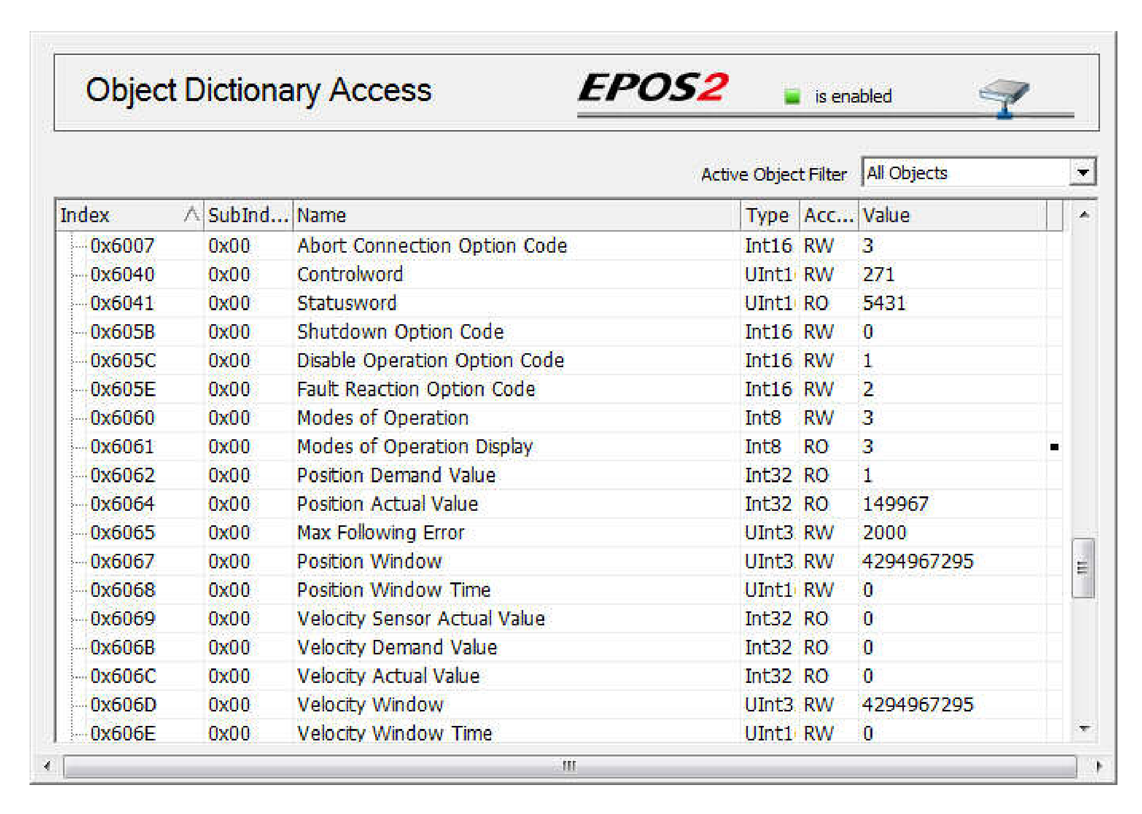
Wichtig festzuhalten ist, dass der Standard in dem Sinne *offen* ist, als es ein gewisses Mass an Freiheit gibt, welche der Objekte und der Funktionalitäten vorhanden sein müssen. Damit ein Gerät dem Standard entspricht, muss es gewisse Objekte enthalten, aber nicht alle. Und der Hersteller ist frei, zusätzliche Objekte und Funktionalitäten hinzuzufügen.

6.4 Das Object Dictionary Tool der EPOS2 [internal]

Lernziele	Systemparameter im Objektverzeichnis editieren. Einen Objektfilter erzeugen.

Nehmen wir als Beispiel das Objektverzeichnis der *EPOS2 [internal]*. Der Motion Control Anwendungsteil benützt Angaben aus dem Objektverzeichnis (z.B. die Zielposition) als Eingabe, um eine Bewegung auszuführen, und schreibt das Resultat (z.B. Zielposition erreicht) in dasselbe Objektverzeichnis.

Auf das Objektverzeichnis der *EPOS2 [internal]* kann über ein *Tool* im *EPOS Studio* zugegriffen werden. Öffne *Object Dictionary* und setze oben rechts den *Active Object Filter* auf *All Objects*.



Index	SubInd...	Name	Type	Acc...	Value
0x6007	0x00	Abort Connection Option Code	Int16	RW	3
0x6040	0x00	Controlword	UInt1	RW	271
0x6041	0x00	Statusword	UInt1	RO	5431
0x605B	0x00	Shutdown Option Code	Int16	RW	0
0x605C	0x00	Disable Operation Option Code	Int16	RW	1
0x605E	0x00	Fault Reaction Option Code	Int16	RW	2
0x6060	0x00	Modes of Operation	Int8	RW	3
0x6061	0x00	Modes of Operation Display	Int8	RO	3
0x6062	0x00	Position Demand Value	Int32	RO	1
0x6064	0x00	Position Actual Value	Int32	RO	149967
0x6065	0x00	Max Following Error	UInt3	RW	2000
0x6067	0x00	Position Window	UInt3	RW	4294967295
0x6068	0x00	Position Window Time	UInt1	RW	0
0x6069	0x00	Velocity Sensor Actual Value	Int32	RO	0
0x606B	0x00	Velocity Demand Value	Int32	RO	0
0x606C	0x00	Velocity Actual Value	Int32	RO	0
0x606D	0x00	Velocity Window	UInt3	RW	4294967295
0x606E	0x00	Velocity Window Time	UInt1	RW	0

Abbildung 40: Objektverzeichnis des Motion Controller EPOS2.
Gezeigt sind die ersten Einträge des Standard Motion Control Profils.

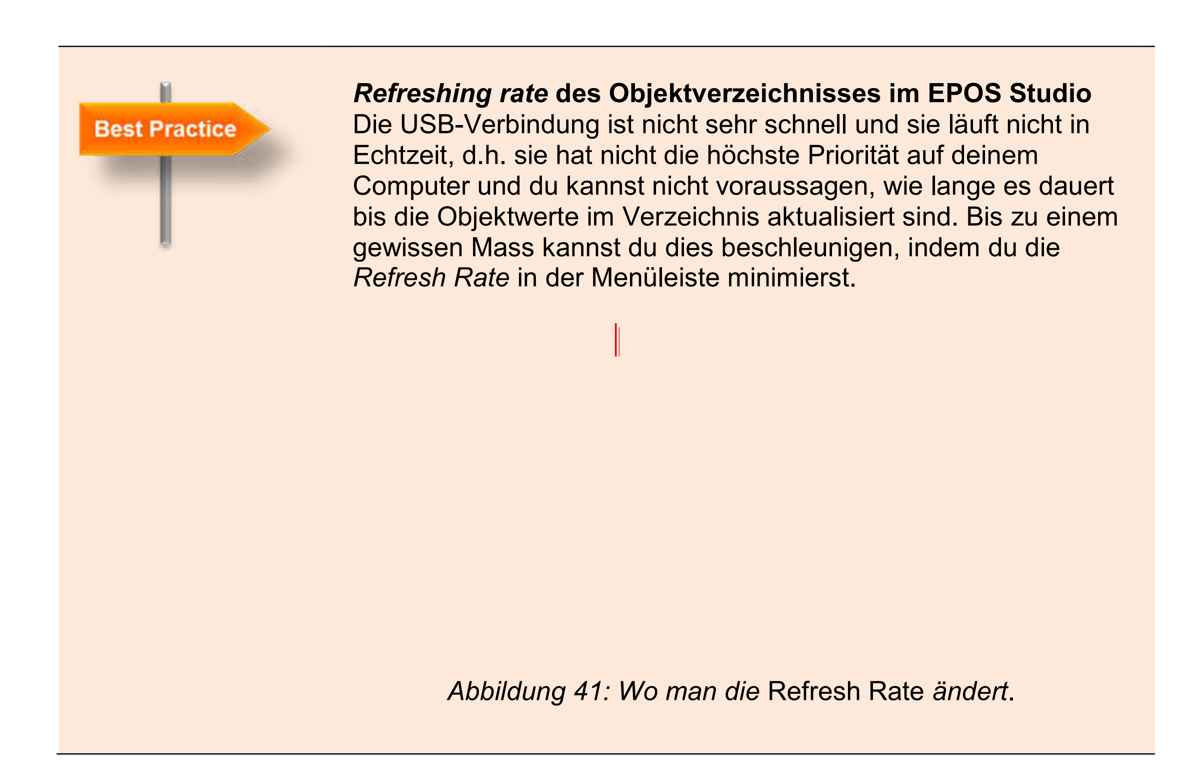
Scrollen wir durch die Liste, finden wir die folgenden Gruppen von Einträgen (Index und Subindex sind in hexadezimalen Nummernsystem angegeben, deshalb das Prefix 0x):

- Index 0x1000 bis 0x1FFF Kommunikationsprofil (im CANopen-Standard)
- Index 0x2000 bis 0x5FFF maxon EPOS spezifisch (nicht im CANopen-Standard)
- Index 0x6000 bis 0x9FFF Standard-Geräteprofil (im CANopen-Standard)

Die maxon spezifischen Einträge enthalten unter anderem die Objekte für nicht-CANopen Kommunikation und die Objekte für die Konfiguration der digitalen und analogen Ein- und

Ausgänge.
Im Standard-Geräteprofil findest du alle Informationen, die Motion Control betreffen.

Das *Tool Object Dictionary* erlaubt den Zugriff auf die gesamte Objektliste. Du kannst jeden schreibbaren Wert ändern. Doppelklicken auf das Objekt genügt oder du benützt das Kontextmenu (rechte Maustaste).



***Refreshing rate* des Objektverzeichnisses im EPOS Studio**
Die USB-Verbindung ist nicht sehr schnell und sie läuft nicht in Echtzeit, d.h. sie hat nicht die höchste Priorität auf deinem Computer und du kannst nicht voraussagen, wie lange es dauert bis die Objektwerte im Verzeichnis aktualisiert sind. Bis zu einem gewissen Mass kannst du dies beschleunigen, indem du die *Refresh Rate* in der Menüleiste minimierst.

Abbildung 41: Wo man die Refresh Rate *ändert*.



Deinen eigenen Objektfilter erzeugen
Den Überblick über das ganze Objektverzeichnis mit seinen vielen Einträgen zu behalten, kann ziemlich herausfordernd sein. Oft interessieren ja nur wenige Objekte. Zu diesem Zweck kannst du deinen eigenen Objektfilter erzeugen, ähnlich zum vordefinierten Filter *System Parameter*.
Hier das Rezept um die PID-Regelparameter zu betrachten:

Schritt 1: Rechtsklick auf das Dialogfenster und wähle *Define ObjectFilter*.
Schritt 2: Wähle *New*, um einen neuen Filter zu erzeugen.
Schritt 3: Gib dem Objektfilter einen Namen, z.B. *MyRegGains*.
Schritt 4: Füge die folgenden Objekte hinzu (Schaltfläche *Add*):

- *0x60F6 (Current Control Parameter Set)*
- *0x60F9 (Velocity Control Parameter Set)*
- *0x60FB (Position Control Parameter Set)*

Schritt 5: *Save* den Objektfilter und *Exit*.



Parameter speichern (*Save All Parameters*)
Änderungen in den Objektverzeichnissen werden erst dann permanent in der EPOS gespeichert, wenn der Befehl *Save All Parameters* explizit ausgeführt wurde (im Kontextmenü, mittels rechter Maustaste auf dem Objektverzeichnis-Dialogfenster).

Eigentlich gibt es drei Sets der Objektverzeichnisse:

- Das *aktuelle Parameterset* im Arbeitsspeicher (RAM), das nach einem Stromausfall verloren ist.
- Das *permanente Parameterset,* das im EEPROM gespeichert ist und nach dem Einschalten aktiv ist.
- Die *default Parameter* (d.h. die Werkeinstellungen) verbleiben immer im EEPROM als letzte Rückfallebene.

6.5 Das Objektverzeichnis der EPOS2 P

Wie Abbildung 38 zeigt, hat der SPS-Teil der *EPOS2 P* drei verschiedene Objektverzeichnisse, für jedes CAN-Netzwerk eines (vgl. auch Abbildung 37).

Neben der üblichen Angaben zur CAN Kommunikation und zu den Geräten enthalten die Objektverzeichnisse der SPS im Wesentlichen die Ein- und Ausgänge für die Prozessvariablen. Mit diesen Daten arbeitet das SPS-Anwenderprogramm. Es benützt die Informationen der Objektverzeichnisse als Eingangsvariablen um beispielsweise eine Bewegung auszuführen – z.B. das Signal aus einem *Position Compare* Ausgang einer Achse, dass eine vorbestimmte Position erreicht worden ist. Die SPS schreibt die Ergebnisse der aktuellen Programmausführung in die entsprechenden Objektverzeichnisse – z.B. ein Ausgangssignal, das zum Stoppen einer Achse benötigt wird.

Allerdings verwenden wir diese Prozessvariablen praktisch nicht für unser SPS-Programm in den folgenden Kapiteln. Stattdessen lesen und schreiben wir die relevanten Angaben direkt in und aus den Slave-Objektverzeichnisse.



SPS Hintergrundinformation
SPS steht für speicherprogrammierbare Steuerung (englisch PLC = *Programmable Logic Controller)*. Im Wesentlichen ist dies ein Rechner (oder besser eine zentrale Recheneinheit) mit einem oder mehreren Programmen, die darauf laufen. Die SPS enthält ein Eingangsmodul, wo Informationen des Prozesses (von Sensoren) eingelesen werden, und ein Ausgangsmodul, wo die Prozessausgaben (für Aktoren und Antriebe) geschrieben werden.

SPS-Programme laufen zyklisch ab: Sie lesen die aktuellen Prozesseingänge, führen damit einen Programmdurchlauf aus und setzen die neuen Prozessausgänge. Dieser Zyklus wird gemäss den *Task*-Einstellungen wiederholt (vgl. Seite 96). Ein Programmzyklus kann beliebig lange dauern, liegt aber meist im Bereich einiger Millisekunden.

6.6 Parameter Up- und Download

Das Objektverzeichnis enthält alle Parameter, die ein CANopen-Gerät beschreiben. Es ist quasi der Fingerabdruck des Geräts. Die Parameterliste kann in einem speziellen elektronischen Datenblatt (*electronic data sheets*), *Device Configuration File* genannt, gespeichert werden (*Parameter Export*). Somit ist es möglich, mehrere Geräte genau gleich zu konfigurieren, einfach indem man das entsprechende elektronische Datenblatt in die *EPOS2 P* hinunterlädt (*Parameter Import*).

Im *EPOS Studio* hat es zu diesem Zweck *Parameter Export/Import* Wizards . Diese erlauben auch das Zurücksetzen auf die Werkeinstellungen (*Default Parameter*).

6.7 Kommunikation

Die Kommunikation in den unteren CAN-Ebenen beruht auf Broadcast-Kommunikation: Jeder Knoten kann eine Nachricht senden, die von jedem anderen Knoten aufgenommen werden kann. Um Datenkollisionen zu vermeiden, hat jedes Telegramm eine definierte Priorität. Starten zwei Nachrichten zur selben Zeit, gewinnt diejenige mit dem tieferen Prioritätswert und wird zuerst übertragen.

In CANopen ist die Kommunikation in dem Sinne spezifischer geregelt, als dass vordefinierte Kommunikationskanäle (jeder mit einem eindeutigen Prioritätswert) beschreiben, welcher Knoten der Sender und welcher der Empfänger einer Nachricht ist (*peer to peer* Kommunikation).

Netzwerk-Management und spezielle Nachrichten

Die tiefsten Prioritätswerte (= höchste Priorität) werden denjenigen Nachrichten zugeordnet, die sicherstellen, dass das Netzwerk korrekt arbeitet und die Kommunikation verlässlich ist. In dieser Kategorie finden wir Nachrichten zur Fehlerbehandlung (*error control*) und Notfallnachrichten (*emergency*), zur Synchronisation und Zeitstempel, als auch Netzwerk-Management-Nachrichten z.B. für das Booten und Ändern des Betriebszustands eines Knotens. Die genaue Priorität all dieser Nachrichten braucht uns hier nicht zu interessieren.

Prozess-Daten-Objekte (PDO)

Die nächsttiefere Priorität wird den Prozessdaten (*Process Data Objects,* PDO) zugeordnet. Mittels PDO wird der Echtzeit-Datenaustausch für kleine Objekte organisiert. Typisch sind dies Daten, die während des Prozesses oft ändern (sic der Name!) und in Echtzeitanwendungen regelmässig aufdatiert werden müssen.
Um die Übertragung zu beschleunigen, werden die Telegramme klein gehalten, indem unnütze Ballastinformation vermieden wird. Die korrekte Übertragung eines PDO-Telegramms wird nicht bestätigt, um die Auslastung des Busses tief zu halten.
Der Inhalt jeder PDO-Nachricht wird im Voraus festgelegt, ebenso Ausgangs- und Zielpunkte (welches Objekt von welchem Knoten). Diese Konfiguration wird *Mapping* genannt, das gleichzeitig auch festlegt, wie oft eine PDO-Nachricht gesendet wird: periodisch oder über ein Ereignis getriggert. Das PDO-Mapping kann in den Objektverzeichnissen der beteiligten Geräte eingesehen werden.

Die CAN PDO-Kommunikation wird in der Regel dazu verwendet, die Prozesseingänge der SPS zu aktualisieren. Es kann mehrere PDO-Kommunikationskanäle zwischen zwei Geräten geben. Beachte, dass die Zykluszeit des SPS-*Tasks* und die Update-Rate der PDO-Daten zwei verschiedene Dinge sind. In gewissen Anwendungen müssen diese synchronisiert werden.

Bemerkung: Alle Befehle, die vom EPOS Studio aus gesendet werden, basieren im Wesentlichen auf SDO-Kommunikation, ebenso die SPS-Programmierung im nächsten Kapitel. Konsultiere das *Programming Reference* Dokument (Kapitel 4.4.2 *Communication via Network Variables*), um mehr über das PDO-*Mapping* in EPOS Systemen zu erfahren.

Service-Daten-Objekte (SDO)

Die tiefste Priorität – die höchsten Prioritätswerte – ist den Service-Daten-Objekten (*Service Data Objects,* SDO) zugeordnet. SDO erlaubt die Datenübertragung beliebiger Grösse. Grosse Nachrichten (> 4 Bytes) werden automatisch segmentiert, d.h. auf mehrere Telegramme aufgeteilt und nacheinander gesendet.

Wie die PDO-Kommunikation ist auch die SDO-Übertragung zwischen zwei spezifischen Knoten definiert. Es werden zwei Kommunikationskanäle benötigt, je einen für jede Richtung. Die Telegramme enthalten die Angaben, welcher Eintrag im Objektverzeichnis gelesen oder beschrieben wird. Die Hälfte des nutzbaren Inhalts wird durch diesen „Ballast“ aufgebraucht.

SDO ist eine relative langsame Kommunikation und ist primär für die Gerätekonfiguration gedacht. Da aber typische Achsbewegungen einige hundert Millisekunden dauern, genügt die SDO Kommunikation für viele Motion Control Anwendungen. Wenn allerdings mehrere Achsen eng koordiniert oder synchronisiert werden müssen, muss dies nicht mehr zutreffen.



PDO und SDO in EPOS-Systemen
PDO Kanäle in EPOS-Systemen sind nur zwischen Slaves und dem Master eingerichtet. Zwischen Slaves gibt es keine vordefinierten Kommunikationskanäle.

- Es hat einen SDO-Kanal in jeder Richtung zwischen der *EPOS2 P* und der *EPOS2 [internal]*
- Es hat 4 PDO-Kanäle zum Senden und 4 PDO-Kanäle zum Empfangen von Prozessdaten in der *EPOS2 [internal]*
- Es hat 32 PDO-Kanäle zum Senden und 32 PDO Kanäle zum Empfangen von Prozessdaten auf dem *EPOS2 P* CAN Slave-Netzwerk, was bis zu 32 Achsen im Netzwerk erlaubt.

Teil 3: Die SPS (Speicherprogrammierbare Steuerung)

In Teil 2 haben wir mit Hilfe der *EPOS Studio* Software den internen Motion Controller erforscht, die *EPOS2 [internal]*. Das *EPOS Studio* eignet sich für das Einrichten und Konfigurieren des Systems. Dies geschieht durch Einzelbefehle, die einer nach dem anderen angewandt werden. Es besteht keine Möglichkeit eine Befehlsfolge zu senden und der EPOS2 Motion Controller hat keinen Speicher für eine solche Sequenz. Die EPOS2 ist, was wir als *online kommandiert* bezeichnen. Einzelne Bewegungs- und I/O-Befehle werden vom übergeordneten System (Master) gesendet und sofort im Slave-Controller ausgeführt. Bis zu diesem Punkt diente das EPOS Studio als Master.

Der Master, den wir nun benützen wollen, ist die eingebaute SPS. In der SPS übernimmt das Anwendungsprogramm die Prozesssteuerung, d.h. die Koordination aller Aktionen der verschiedenen Slaves. Das Programm liest die Information von Eingängen, sendet Bewegungsbefehle und setzt Ausgänge. Die Aufgabe in diesem dritten Teil des Buches ist das Schreiben eines einfachen Programms zur Prozesskontrolle.

- In Kapitel 7 werfen wir einen ersten Blick auf das Programmierwerkzeug und den SPS-Programmierstandard IEC 61131-3.
- In Kapitel 8 schreiben wir unser erstes kleines SPS-Programm.
- Kapitel 9 vertieft unser Wissen, wie die zusätzlichen Ein- und Ausgänge verwendet werden können, z.B. für eine Referenzfahrt.
- In Kapitel 10 programmieren wir einen Funktionsbaustein selber und fügen ihn ins SPS-Programm ein.

7 Das Programmierwerkzeug starten

Ziel dieses Kapitels ist es, einen ersten Eindruck vom Software-Werkzeug zu gewinnen, mit dem wir SPS-Programme schreiben. Das Werkzeug heisst *OpenPCS®* und stammt von der deutschen Firma *infoteam®*. *OpenPCS* ist ein Editor zur Erzeugung von Programmen gemäss dem internationalen SPS-Programmierstandard *IEC 61131-3* , dem weltweit am weitesten verbreiteten SPS-Programmierstandard. IEC 61131-3 wird von der Organisation *PLCopen* entwickelt, unterhalten und gefördert. Wenn du diesen Standard kennst, musst du nur noch den *OpenPCS* Editor bedienen lernen; wenn nicht, wirst du die Grundlagen in den folgenden Kapiteln lernen.

EPOS-Systeme sind CANopen-Geräte, und als erster Schritt muss das CANopen-Netzwerk mit all seinen Eigenschaften im *EPOS Studio* eingerichtet werden. Erst dann sollten wir zur SPS-Programmierung schreiten. Somit öffne das *OpenPCS* aus dem *EPOS Studio* (und nicht separat), um sicherzustellen, dass die gesamten relevanten Informationen der beteiligten Geräte und des CANopen-Netzwerks für die *EPOS2 P* SPS-Programmierung zur Verfügung stehen. Beispielsweise müssen Achsnummern, Konfiguration der Ein- und Ausgänge und Prozessdatenobjekte (PDO) bekannt sein.

Für dieses Kapitel brauchen wir keine besonderen Einstellungen; die Standardeinstellungen im Objektverzeichnis des EPOS2 Motion Controller passen, sogar mit den Modifikationen, die wir in den vorausgehenden Kapiteln vorgenommen haben. Stelle nur sicher, dass nicht zufällig ein Eingang aktiviert ist, der eine Fehlermeldung erzeugt, z.B. ein Endschalter.

Somit bleibt uns nur, das *IEC 61131 Programming Tool* der *EPOS2 P* zu starten.

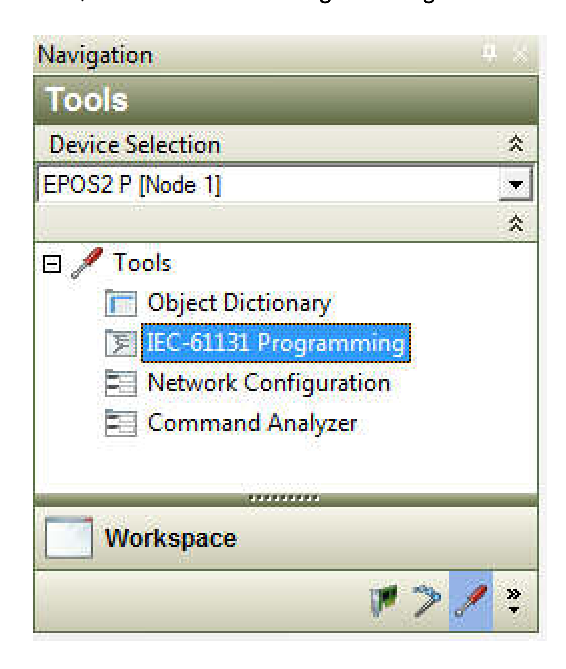


Abbildung 42: Das IEC-61131 Programming Tool *im* EPOS Studio *öffnen.*

7.1 Sample Project: Simple Motion Sequence

Bevor wir selber programmieren, betrachten wir ein bereits bestehendes Programm, ein Projekt namens *SimpleMotionSequence*. Aktiviere dazu *SimpleMotionSequence* in der *Sample Project* Liste und öffne es rechts durch Betätigen der Schaltfläche *Open Sample Project*.

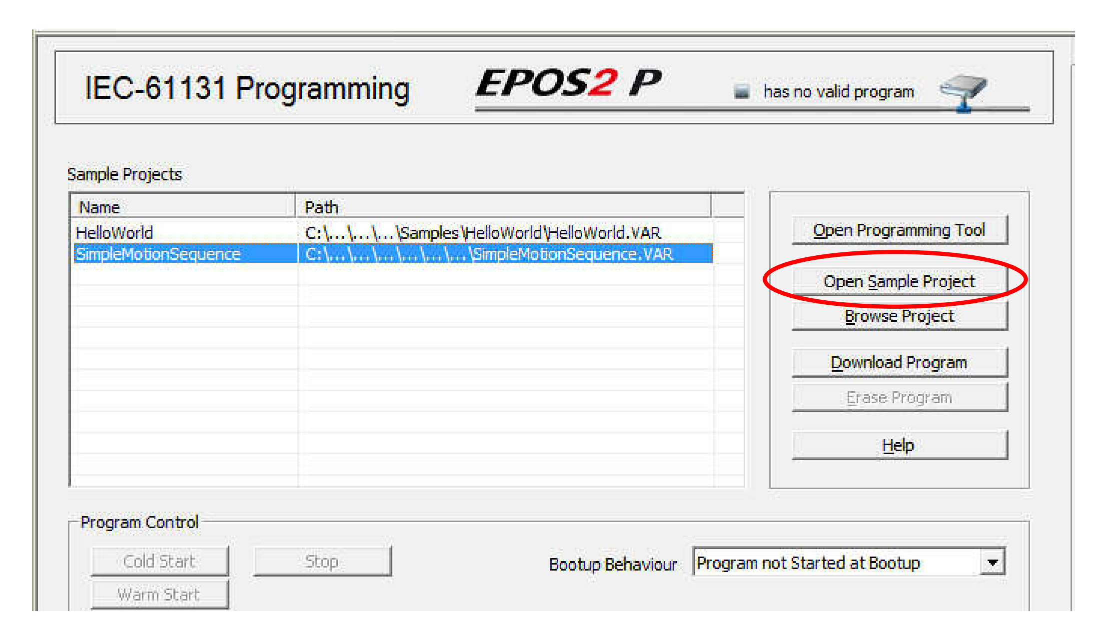


Abbildung 43: Öffnen des OpenPCS Tool *mit einem bestehenden* Sample Project.
Aktiviere das entsprechende Projekt in der Liste und klicke rechts auf die Schaltfläche Open Sample Project.

Die *OpenPCS* Software startet in einem neuen Fenster und zeigt die Dateistruktur des Projekts im linken Teil des Fensters.



Blaue LED: Anzeige des SPS-Programmstatus
Die blaue LED auf der *EPOS2 P* zeigt den Status des Programms in der SPS an.

- Schnell blinkend — Kein Programm in der SPS enthalten
- Langsam blinkend — Ein Programm ist in der SPS geladen, aber nicht gestartet
- Dauernd leuchtend — Ein Programm läuft

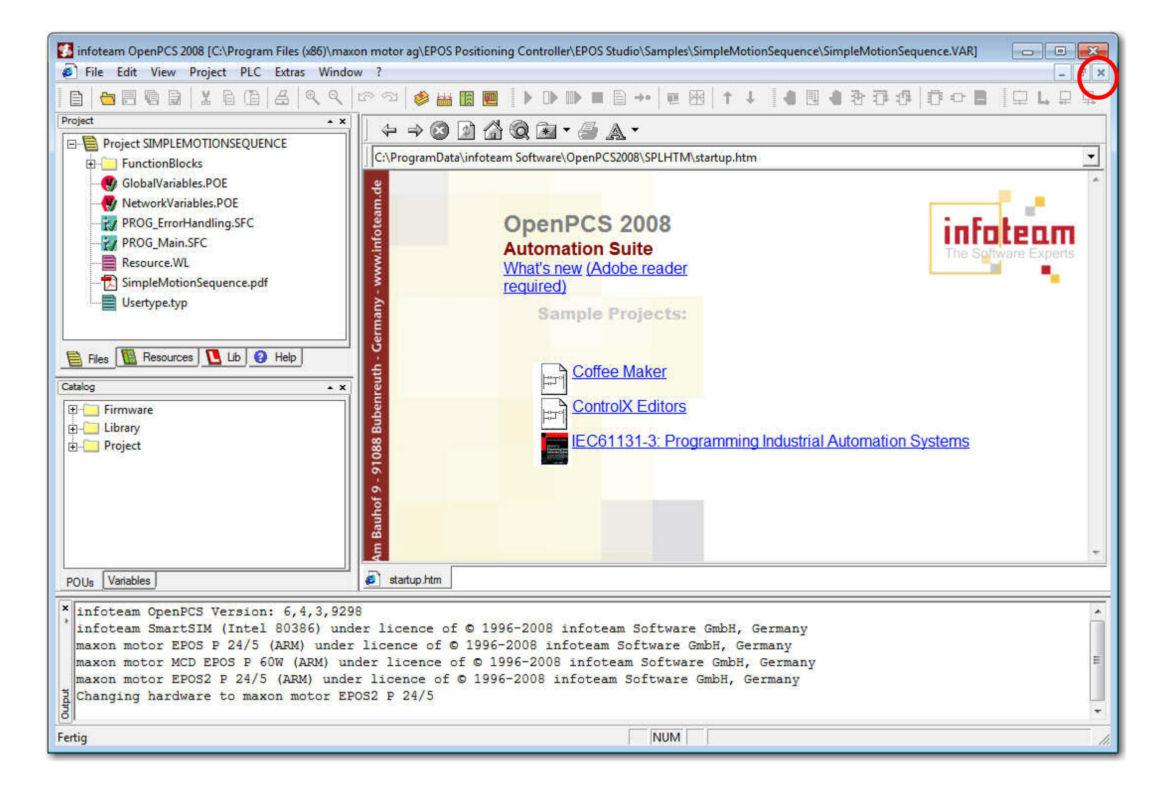


Abbildung 44: Der Bildschirm des SimpleMotionSequence *Projekts. Die Dateistruktur befindet sich links oben. Das Hauptfenster zeigt den* OpenPCS-*Startbildschirm, der nicht weiter benötigt wird und am besten durch klicken auf den kleinen x Knopf oben rechts geschlossen wird (nicht das Programm!).*

Bevor wir die verschiedenen Dateien analysieren, starten wir das Programm, um zu sehen was es tut. Folge einfach diesen Schritten:

Schritt 1 Kompilieren oder *Build Active Ressource*: Klicke auf den entsprechenden Befehl im *PLC* Menü oder den entsprechenden Kurzbefehl.

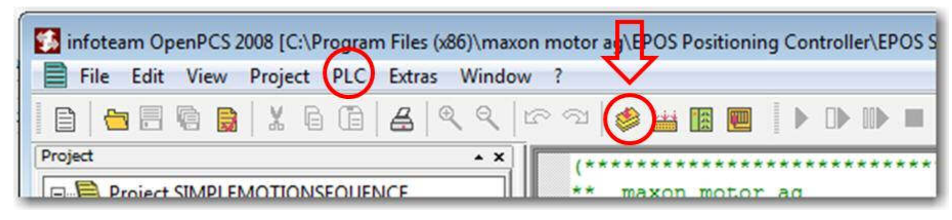


Abbildung 45: Build Active Ressource

Das Resultat der Kompilation sollte ein *0 error(s) 0 warning(s)* in der letzten Zeile des Ausgabefeldes (*Output*) unten auf dem Bildschirm sein.

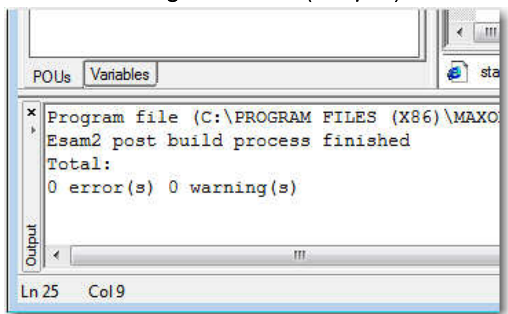


Abbildung 46: Errors *und* Warnings *des Kompiliervorgangs*

Schritt 2 Verbindung zur SPS aufbauen und das Programm auf die SPS laden.
Klicke auf die *Go Online/Offline* Schaltfläche im *PLC* Menü oder auf den entsprechenden Kurzbefehl. Bestätige die Nachricht, dass du das Programm auf die SPS runterladen willst.

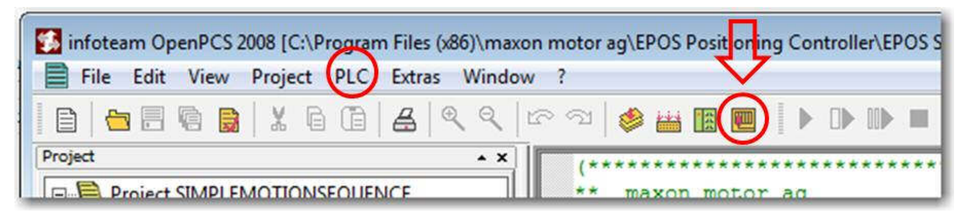


Abbildung 47: Verbindung zur SPS. Go Online/Offline.

Schritt 3 Das Programm starten.
Klicke auf die *Coldstart* Schaltfläche im *PLC* Menü oder auf den entsprechenden Kurzbefehl.

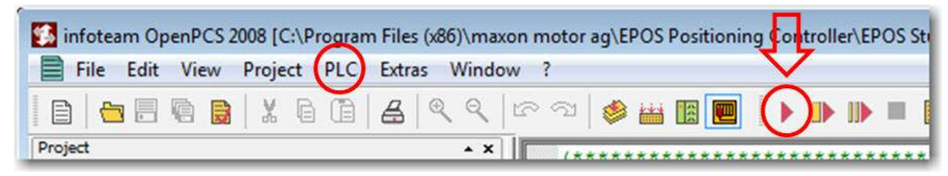


Abbildung 48: Coldstart *des SPS Programms*

Schritt 4 Beobachte, was der Motor tut.
Nach ein paar Sekunden dreht der Motor langsam gegen den Uhrzeigersinn (CCW) gefolgt von einer schnellen Bewegung in Uhrzeigerrichtung (CW). Anschliessend folgt eine kurze Pause und die gesamte Bewegungssequenz wird laufend wiederholt.

Schritt 5 Das Programm stoppen.
Klicke auf *Stop* im *PLC* Menü oder auf den entsprechenden Kurzbefehl.



Das PLC Menü und die Kurzbefehle im OpenPCS
Das *PLC* Menü enthält alle Befehle im Zusammenhang mit der SPS der *EPOS2 P*, wie eine Verbindung herstellen (*Go Online/Offline*), das Programm kompilieren (*Build Active Resource*), Programme hinauf- und herunterladen, starten und stoppen, sowie ihre Ausführung überwachen.
Für die meisten Funktionen gibt es Schaltflächen mit Kurzbefehlen unterhalb der Menüleiste.

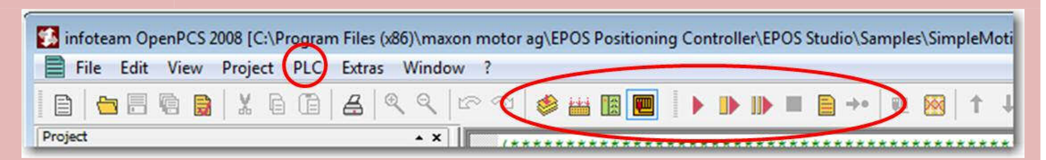


Abbildung 49: Kurzbefehle im OpenPCS.

7.2 Die Projektdateien

Abbildung 50 zeigt die Dateistruktur des *SimpleMotionSequence* Projekts. Die einzelnen Dateitypen werden durch spezielle Symbole dargestellt.

Kurz gesagt, es gibt Dateien, die Variablen oder Variablentypen deklarieren, und die im gesamten Projekt Verwendung finden: *GlobalVariables.POE*, *NetworkVariables.POE* und *USERTYPE.TYP*. Es hat zwei Programme (*PROG_ErrorHandling.SFC* und *PROG_Main.SFC*), welche benutzerdefinierte Funktionsbausteine (*Function Blocks*) verwenden und die im entsprechenden Verzeichnis abgelegt sind. Eine Projektbeschreibung ist in *SimpleMotionSequence.pdf* enthalten. Zusätzlich gibt es seine Datei namens *Resource.WL*, welche zum Überwachen der Variablen im Betrieb dient (WL steht für *Watch List*).

Mit Ausnahme von *Resource.WL* können die Dateien einfach mit einem Doppelklick geöffnet und im Hauptfenster dargestellt werden.

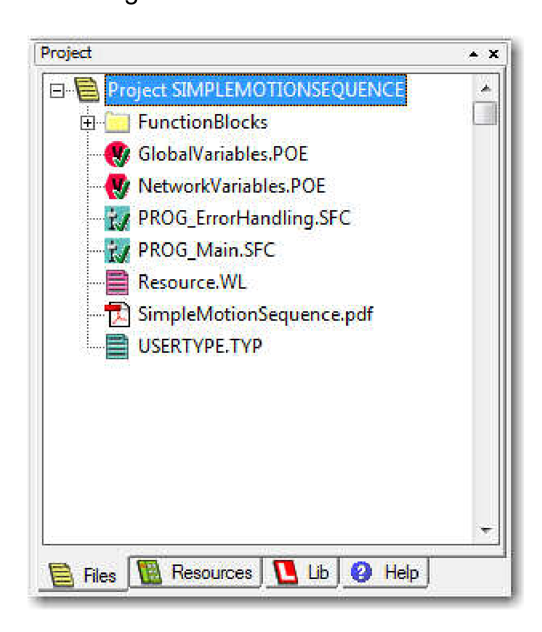


Abbildung 50: Das Fenster mit den Projektdateien.

SimpleMotionSequence.pdf

Schauen wir uns zuerst *SimpleMotionSequence.pdf* an. Die Datei liefert eine Projektbeschreibung und zusätzliche Übungen, um Variablen zu überwachen.

Abschnitt 3.1 enthält die Kerninformation über das Projekt. Die darin enthaltenen Tabellen und das Diagramm verraten uns die Struktur des Hauptprogramms (*PROG_Main.SFC*), welches als Zustandsmaschine (*State machine*) aufgebaut ist. Der Einfachheit halber lassen wir das Fehler-Handlingsprogramm (*PROG_ErrorHandling.SFC*) auf der Seite, ebenso wie die Error-Handling-States im Hauptprogramm. Die restlichen für die eigentliche Antriebsaufgabe relevanten Zustände (States) ergeben folgendes Szenario.

Abbildung 51: Die State Machine *des* SimpleMotionSequence *Projekts Angepasste Darstellung mit Hervorhebung der wichtigsten Zustände und den Wartebedingungen für die Übergänge dazwischen. Es hat eine 5s Pause nach der Initialisierung und 2s nach jeder Positioniersequenz.*

Nach dem Programmstart – durch den schwarzen Punkt symbolisiert – wird die Achse im Initialisierungs-State (*Init*) zurückgesetzt, d.h. Fehler werden gelöscht, wie wir später sehen werden. Nach einer Pause von 5 Sekunden erfolgt der Übergang (*Transition*) zum nächsten State (*Standstill*). *Standstill* setzt die Achse auf *Enable* und wartet 2 Sekunden, bevor die Transition zum State *Position Sequence* erfolgt. In *Position Sequence* werden die beiden Bewegungen in CCW und CW Richtung ausgeführt, gefolgt von einer Transition zurück zum State *Standstill*.

Im Kapitel 4 des *SimpleMotionSequence.pdf* Dokuments findest du weitere Angaben, wie du den Variablenwerten und den States on-line folgen kannst und wie du einen Fehler provozieren und das Error Handling beobachten kannst. Dies sind aber Themen, die wir hier nicht weiter vertiefen.

Eine Bemerkung bevor wir die anderen Dateien genauer anschauen: *SimpleMotionSequence* ist sehr strikt programmiert und um die Programmstruktur und die Hierarchieebenen zu verstehen, muss man sich etwas reinknien. Dies gilt vor allem für Personen die sich in der SPS-Programmierung mit IEC 61131 nicht auskennen. Das Ziel dieses Kapitels ist aber nicht, in die Details der Programmierung einzutauchen, sondern die Hauptelemente eines SPS-Projekts kennenzulernen.

Programmdateien

Was wir als erstes in der Dateistruktur erkennen, ist, dass ein SPS-Projekt mehr als ein Programm enthalten kann. Wir haben ein Hauptprogramm (*PROG_Main.SFC*) und parallel dazu ein Programm, das im Falle eines Fehlers übernimmt (*PROG_ErrorHandling.SFC*). Beide Programme sind in der Programmiersprache Ablaufsprache (*Sequential Function Chart* daher die Datei-Endung .SFC) geschrieben. Die Ablaufsprache erlaubt es, Programme auf einfache Weise als Zustandsmaschine (*State Machine*) zu strukturieren. In unserem Fall sind die individuellen States oder *Schritte* als *Strukturierter Text (ST)*, einer weiteren Programmiersprache des IEC 61131-3 Standards, programmiert.

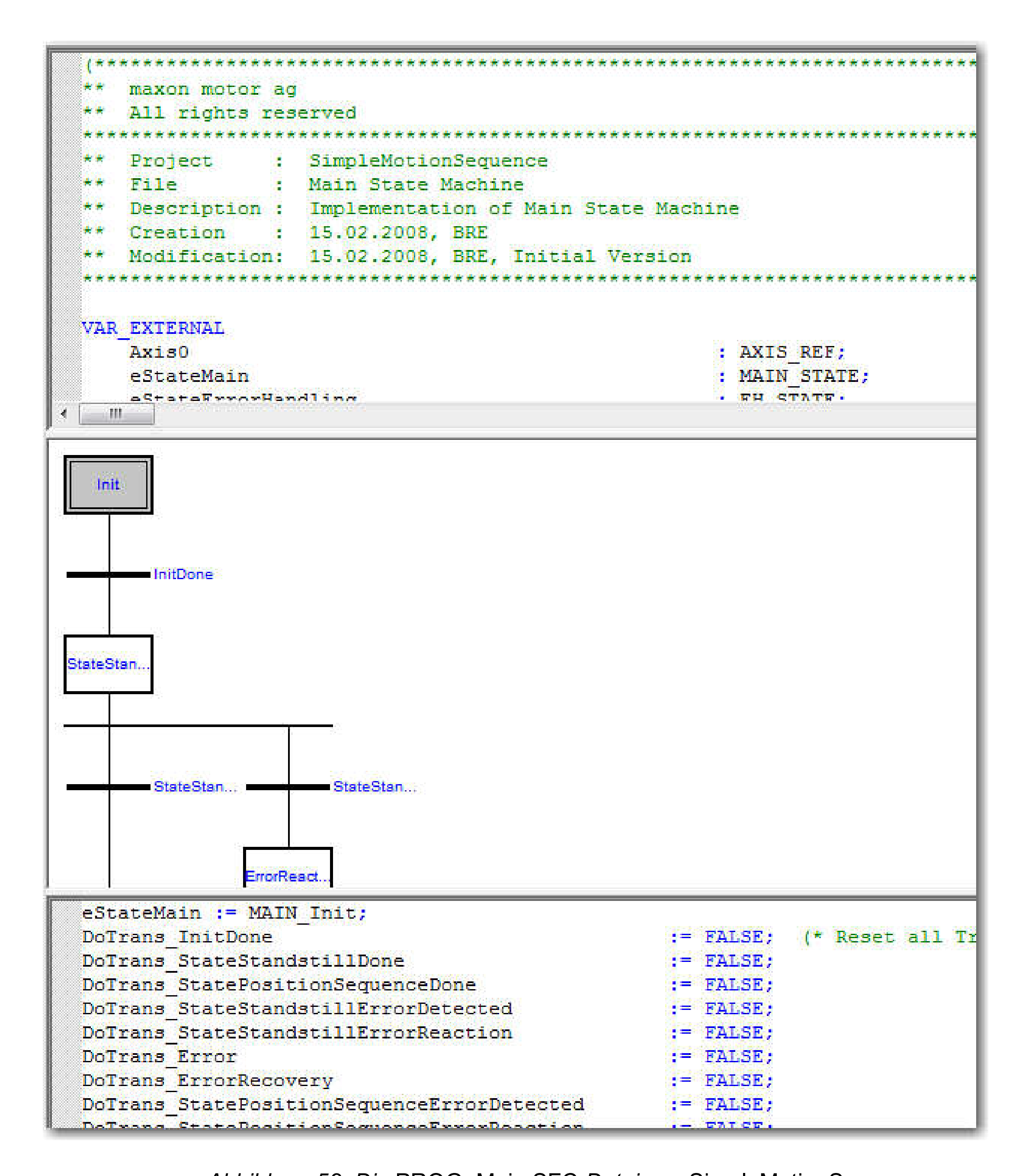


Abbildung 52: Die PROG_Main.SFC *Datei von* SimpleMotionSequence*.*
Oben: Variablendeklaration.
Mitte: Die Programmstruktur in SFC *mit Schritten und Transitionen. Der Initialisierungsschritt ist grau hervorgehoben.*
Unten: Der Programmcode des Initialisierungsschritts in Structured Text*.*

Betrachten wir *PROG_Main.SFC* etwas genauer. Ein Doppelklick öffnet die Datei. Die Abbildung 52 zeigt das Hauptfenster im *OpenPCS*. Im zentralen Teil findet man dieselbe *State Machine* Struktur wie in Abbildung 51. Die Schritte (*Steps*) sind als Kästchen dargestellt und von *Transitionen* getrennt, welche die Bedingung enthalten, wann zum nächsten *Step* fortgeschritten werden kann. Klicken auf einen Schritt oder eine Transition zeigt den zugehörigen Programmcode im unteren Teil des Fensters.

Um die wichtigsten Eigenschaften der Programmierung zu erklären, betrachten wir den *Init* Schritt, welcher in *ST* programmiert ist (Abbildung 53). In den ersten 12 Linien werden Organisationsvariablen zurückgesetzt, damit alle möglichen Transitionen zwischen Schritten ausgeschaltet sind; alle diese Variablen werden auf FALSE gesetzt. Damit ist sichergestellt, dass alles richtig aufgesetzt ist, egal wie man diesen *Init* State erreicht, beispielsweise aus einem Fehlerbehebungszustand.

Die Hauptaktion im *Init* Schritt ist in einem benutzerdefinierten Funktionsbaustein (*Function Block*) namens *fbInit* versteckt. Die letzte IF…END_IF Konstruktion organisiert die Transition zum nächsten Schritt: Die Transition wird ausgeführt (d.h. die Variable *DoTrans_InitDone* wird TRUE), falls *fbInit* korrekt beendet wurde (d.h. falls *fbInit.done* erfüllt ist).

Zusammengefasst: Der Code des *Init* Schritts zeigt einiges an Organisation für die State Machine und einen Funktionsbaustein *fbInit*, in dem die eigentliche Aktion verborgen ist.

```
eStateMain := MAIN_Init;
DoTrans_InitDone                                  := FALSE;
DoTrans_StateStandstillDone                       := FALSE;
DoTrans_StatePositionSequenceDone                 := FALSE;
DoTrans_StateStandstillErrorDetected              := FALSE;
DoTrans_StateStandstillErrorReaction              := FALSE;
DoTrans_Error                                     := FALSE;
DoTrans_ErrorRecovery                             := FALSE;
DoTrans_StatePositionSequenceErrorDetected        := FALSE;
DoTrans_StatePositionSequenceErrorReaction        := FALSE;

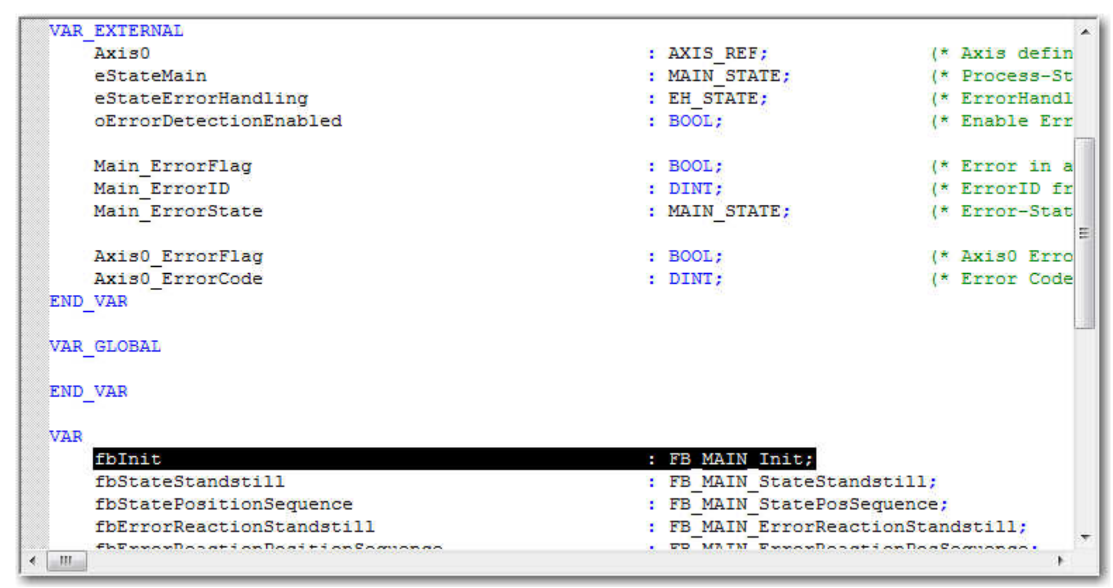
oExecuteStatePositionSequence                     := FALSE;
oExecuteStateStandstill                           := FALSE;

fbInit(Axis := Axis0, Execute := oExecuteInit);
oExecuteInit := TRUE;
IF fbInit.Done THEN
   oExecuteInit := FALSE;
   DoTrans_InitDone := TRUE;
END_IF;
```

Abbildung 53: Programmcode (ohne Kommentare) des Init *Schritts geschrieben in der Programmiersprache* Strukturierter Text *(ST).*

*Funktionsbausteine (*Function Blocks*)*

Wo finden wir Detailangaben zu *fbInit*? Da dies ein Funktionsbaustein (mit Variablen) ist, muss er wie eine Variable deklariert werden. In der Variablendeklaration erkennen wir, dass *fbInit* eine Instanz des Funktionsbausteins *FB_MAIN_Init* ist (Abbildung 54). Die Deklaration und Definition des letzteren ist neben anderen im Verzeichnis *FunctionBlocks* der Dateistruktur aufgeführt.



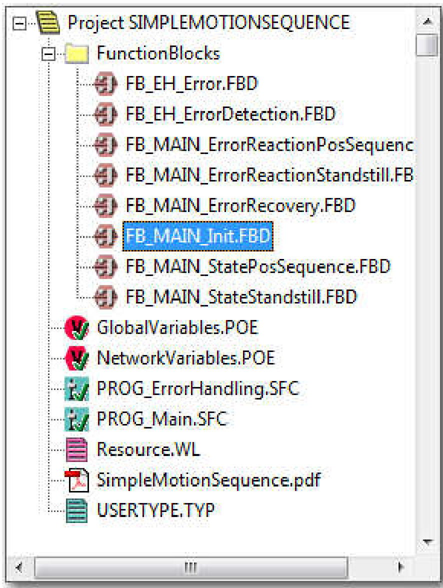


Abbildung 54: fbInit
Oben: Deklaration von fbInit *als Instanz von* FB_MAIN_Init *in der Variablendeklaration des* PROG_Main.SFC*.*
Unten: Wo man FB_MAIN_Init.FBD *im Dateiverzeichnis findet.*

Die Datei-Extension von *FB_MAIN_Init* ist *FBD*, was für *Function Block Diagram* (Funktionsbausteinsprache*)* steht. *FBD* ist eine graphische Programmiersprache im *IEC 61131-3* Standard. In der geöffneten *FB_MAIN_Init.FBD* Datei ist ersichtlich, dass diese Sprache stark auf das Aneinanderfügen von Funktionsbausteinen abstützt (Abbildung 55). In diesem Beispiel werden vordefinierte Funktionsbausteine verwendet, um einen neuen Funktionsbaustein zu erzeugen. Ein Timer-Funktionsbaustein *TON* (im *IEC 61131-3* Standard enthalten) wird aufgerufen, sobald die Achse erfolgreich zurückgesetzt ist (*Reset*). *MC_Reset* ist ein Standard Motion Control Funktionsbaustein, der die Fehler löscht und die Endstufe sperrt (Disable).

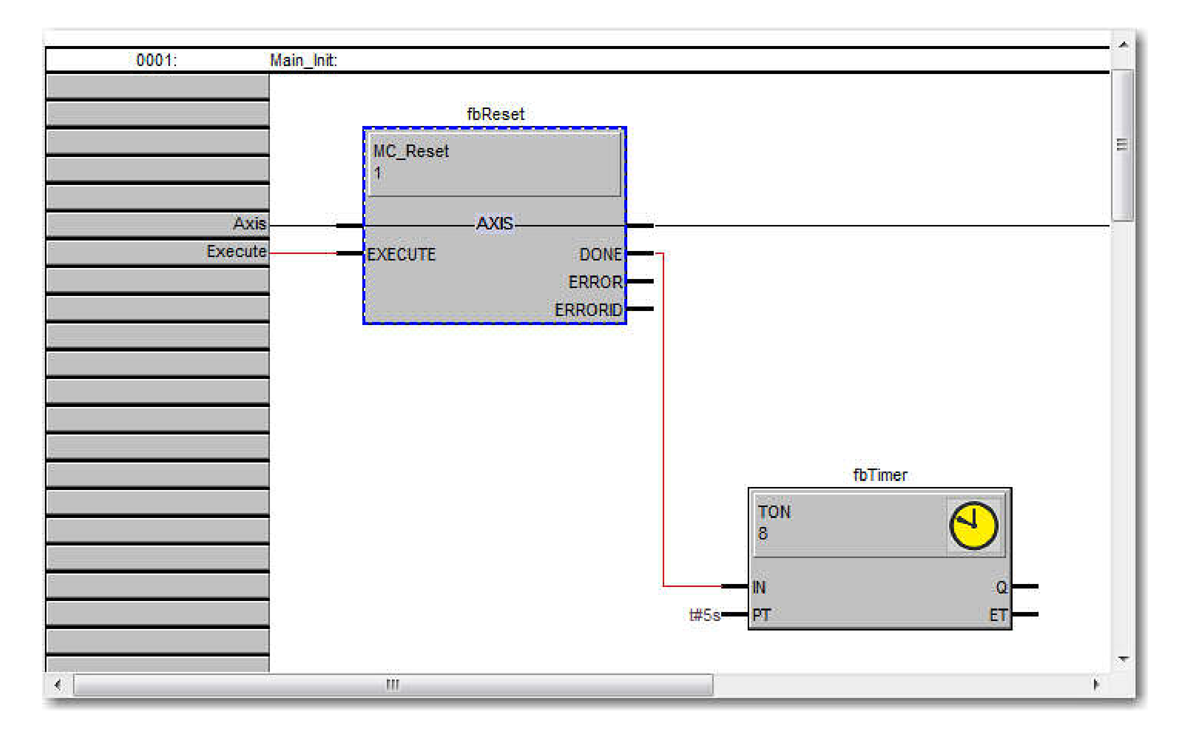


Abbildung 55: Programmcode von FB_MAIN_Init.FBD*.*

Man kann sich Funktionsbausteine als eigenständige Module und Unterprogramme vorstellen. Funktionsbausteine haben Eingangs- und Ausgangsparameter, wie man sehr schön in der graphischen FBD Programmiersprache erkennt (Abbildung 55). *fbInit* ist ein Beispiel eines selbst definierten Funktionsbausteins. Es gibt aber viele vordefinierte Funktionsbausteine, die man ebenfalls verwenden kann. Du findest sie links im *Catalog*-Teil des *OpenPCS*-Bildschirms (Abbildung 56).



Farbcodierung des Programmtextes im *OpenPCS*
Der OpenPCS Editor identifiziert die reservierten Schlüsselwörter, Ausdrücke, vordefinierte Funktionen und Funktionsbausteine aus dem *IEC 61131-3* Standard und hebt sie **blau** hervor.
Grün wird für Kommentare verwendet, welche in Klammern mit Sternchen gesetzt werden müssen, **(* … *)**.

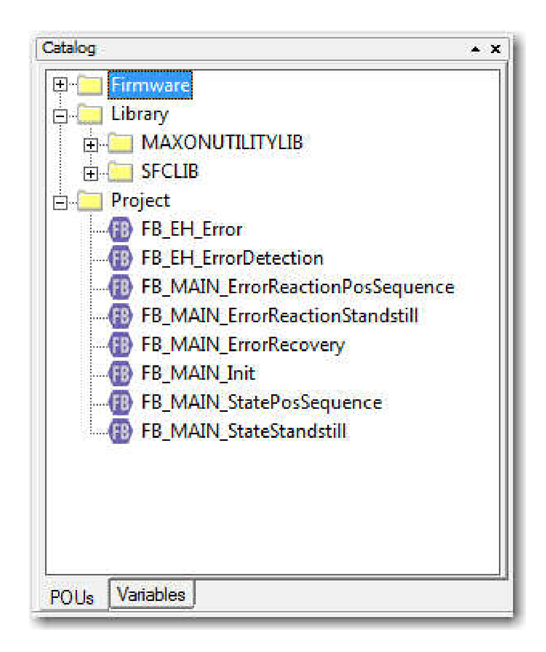


Abbildung 56: Catalog *der verfügbaren Funktionsbausteine (und Funktionen).*

So, damit sind wir am Ende unserer kleinen Tour durch die *OpenPCS* Software. Du solltest nun …

- den Zweck der verschiedenen Dateitypen in einem *OpenPCS* Projekt kennen und verstehen.
- die verschiedenen Dateien eines bestehenden *OpenPCS* Projekts öffnen und betrachten können.
- ein bestehendes SPS Programm in der *EPOS2 P* speichern (*download*), starten und stoppen können.



Variablen

Datentypen

Der *IEC 61131-3* Programmierstandard stützt sich stark auf Variablen ab, die zum Initialisieren, für den Prozess und zum Speichern von Daten benützt werden. Variablen müssen deklariert und einem bestimmten Datentyp wie BYTE oder INTEGER zugeordnet werden. Der Standard beinhaltet auch die Möglichkeit, eigene Typen zu definieren. Zum Beispiel, kann man Arrays und komplexe Strukturen erzeugen oder – wie man in der Datei *Usertype.typ* ersieht – Aufzählungstypen, d.h. Datentypen, bei denen die Variablen nur Werte aus einer endlichen Liste annehmen können.

Startwerte

Man kann den Variablen auch Startwerte zuordnen. Dies sieht dann wie bei diesen beiden Beispielen aus:

- IntVar : DINT := 5;
- BoolVar : BOOL := False;

```
VAR_EXTERNAL

END_VAR

VAR_INPUT
    Execute             : BOOL;
END_VAR

VAR_IN_OUT
    Axis                : Axis_REF;
END_VAR

VAR_OUTPUT
    Done                : BOOL;
    Error               : BOOL;
    ErrorID             : DINT;
END_VAR

VAR
    fbReset             : MC_Reset;      (* Create Instance of MC_Reset *)
    fbTimer             : TON;           (* Create Instance of TON *)
    fbSelection         : MU_Selection;  (* Create Instance of MU_Selection *)
END_VAR
```

Abbildung 57: Die Variablendeklaration des Funktionsbausteins FB_Main_Init.FBD*.*
Es sind keine externen Variablen deklariert. Die Ausführung dieses Funktionsbausteins beginnt mit einer ansteigenden Flanke auf der Input-Variable Execute*. Die Achsnummer wird nach Übereinkunft als Input- und Output-Variable angegeben. Die Beendigung des Funktionsbausteins wird über die Output-Variable* Done *signalisiert. Weitere mögliche Output-Signale sind: Fehlermeldung (*Error*) während der Ausführung und eine dazugehörige Identifikationsnummer (*ErrorID*).*
Der letzte Abschnitt in der Deklaration gehört den lokalen Variablen, hier drei Instanzen von Funktionsbausteinen, die in FB_Main_Init.FBD *verwendet werden.*

Variablendeklaration
Variablen werden in einem separaten Abschnitt oben im Programmierfenster eines Programms oder eines Funktionsbausteins deklariert (Abbildung 52 und oben in Abbildung 54). Sie werden in verschiedene Abschnitte gruppiert, je nach ihrer Verwendung im Projekt.

Variablen, die global verwendet werden – d.h. in mehreren Programmen eines Projekts – müssen in einer eigenen Datei deklariert werden: In unserem *SimpleMotionSequence* Projekt ist dies die Datei *GlobalVariables.POE*. Sie enthält die Achsnummern, die überall im Projekt gültig sind, sowie Variablen, welche die Interaktion zwischen den beiden Programmen steuern. Wenn solche globalen Variablen in einem spezifischen Programm Verwendung finden, müssen sie "importiert", d.h. als externe Variablen deklariert, werden, wie man oben in Abbildung 54 ersieht.

Funktionsbausteine haben typischerweise Schnittstellenvariablen. Das sind Input-Variablen als Startparameter beim Aufruf des Funktionsbausteins und Output-Variablen für die Ergebnisse.

Und dann sind da die lokalen Variablen und Instanzen von anderen verwendeten Funktionsbausteinen. Die lokalen Variablen sind nur innerhalb des spezifischen Funktionsbausteins gültig (vgl. Abbildung 57 als Beispiel).



EPOS Datentyps
EPOS2 P Systeme unterstützen nicht alle elementaren Datentypen, die im *IEC 61131-3* Standard möglich sind. Insbesondere gibt es keine REAL-Typen oder Typen, die auf REAL aufbauen.

Zahlen können nur auf INTEGER basierenden Typen dargestellt werden.

Für Motion Control Anwendungen sind zwei besondere Datentypen vordefiniert:
- *AXIS_REF* bezeichnet eine Achsnummer.
- *MC_Direction* definiert die Drehrichtung bei Drehzahlregelung mit den beiden Werten *MCpositive* und *MCnegative*.

Diese zwei Typen stehen automatisch zur Verfügung (keine Typendefinition nötig), wenn man die *OpenPCS* Software aus dem EPOS Studio startet oder wenn man als *OpenPCS* Projekt eines vom Typ *maxon motor ag EPOS* wählt.

8 Mein erstes Programm: *AxisMotion*

Dieses Kapitel führt dich durch alle notwendigen Schritte, um ein erstes SPS-Programm für die *EPOS2 P* zu erzeugen. Wir programmieren eine Bewegungssequenz für die Achse im Starter Kit. Wie im *SimpleMotionSequence* Beispielprojekt des vorangehenden Kapitels wählen wir *Sequential Function Chart (SFC)* als Programmiersprache, um die allgemeine Programstruktur zu erzeugen. Die einzelnen Schritte führen wir mit *Structured Text (ST)* aus. Wir wenden vordefinierte Funktionsbausteine aus der Standardbibliothek von *IEC 61131-3* (z.B. *TON*) an und weitere aus der Motion Control (MC) Bibliothek: *MC_Reset*, *MC_Power*, *MC_MoveRelative*, *MC_MoveVelocity*, *MC_Stop* ….

Beachte aber, dass unsere Programmierung nur die Bewegung umfasst; ein Errorhandling ist nicht eingeschlossen. In realen Anwendungen muss das Verhalten bei Fehlern berücksichtigt werden – ein unterbrochenes Motor- oder Encoderkabel oder sonst ein Problem kann immer auftreten. Das *SimpleMotionSequence* Beispiel des vorangehenden Kapitels gibt eine Idee, wie ein Errorhandling implementiert werden könnte.



Programming Reference
Das für die Programmierung wichtigste Dokument in der *EPOS Studio* Dateistruktur ist die *Programming Reference*; vgl. dazu Abbildung 4 und Abbildung 5, um es zu finden.
Die *Programming Reference* enthält die Beschreibung der grundlegenden Programmierschritte, der Vorbereitung des Systems und der zusätzlichen Funktionsbausteine, die nicht Teil der *IEC 61131-3* Standardbibliothek sind. Die zusätzlichen Bibliotheken (*libraries*) enthalten Funktionsbausteine für Motion Control (MC), zum einfachen Handling von *EPOS2* Funktionalitäten (MU = *maxon utilities*) und für allgemeine CANopen Befehle, wie zum Lesen und Schreiben von Objekten und zum CAN-Netzwerkmanagement.

Übersicht über das Programm AxisMotion

Die Aufgabe des *AxisMotion* Programms besteht darin, eine Abfolge von Achsbewegungen und Stopps auszuführen. Eine mögliche Sequenz ist Abbildung 58 skizziert.

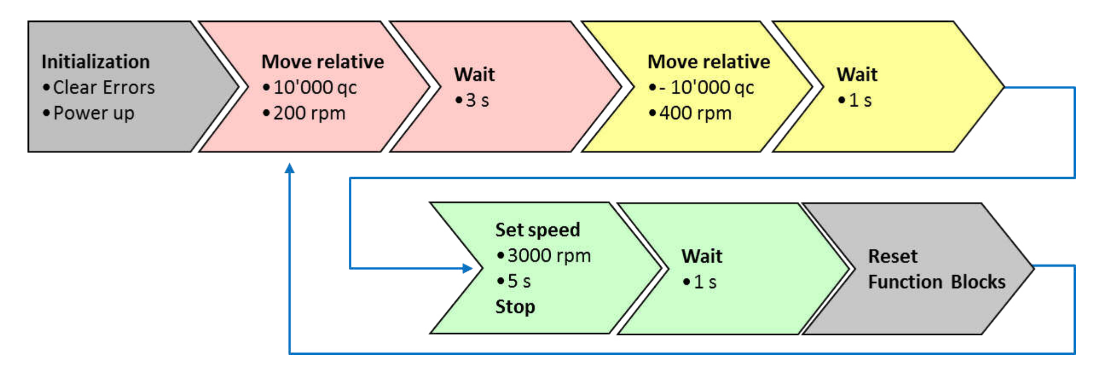


Abbildung 58: Mögliche Abfolge in AxisMotion
Dies dient als Basis zur Programmierung in Sequential Function Chart. *Die verschiedenen Farben kennzeichnen die verschiedenen Etappen der Komplettierung in den nächsten Kapiteln.*

8.1 Vorbereitungsarbeiten

Lernziele	Das System für die Programmierung vorbereiten; inklusive Achsdefinition, Anlegen eines SPS Programmierprojekts und Definition globaler Variablen.

Achsnummer definieren

Die Definition der Achsnummer ist entscheidend, da sich beim Aufruf die meisten Funktionsbausteine in Motion Control auf diese Nummer beziehen. Dieser Abschnitt folgt den Anweisungen, wie sie in der *Programming Reference* (Kapitel 4.3.1) beschrieben sind.

Das Festlegen von Achsnummern ist Teil der Netzwerkkonfiguration (*Network Configuration*) und wird im *EPOS Studio* ausgeführt. Diese Angaben werden später aus dem *EPOS Studio* ins *OpenPCS* übernommen. Schliesse somit zuerst die *OpenPCS* Software und folge diesen Schritten um die Achsnummer der internen *EPOS2* zu definieren.

Schritt 1 Öffne das *EPOS2 P* Tool *Network Configuration*. (Das dauert seine Weile, da einiges an Daten gelesen werden muss.)
Schritt 2 Wähle oben links das Netzwerk *Internal Network CAN-I.*
Schritt 3 Darunter wähle das Device *EPOS2 [internal]*.
Schritt 4 Wähle *Axis Number* als *Axis 1* und *Axis Type* als *Standard*.

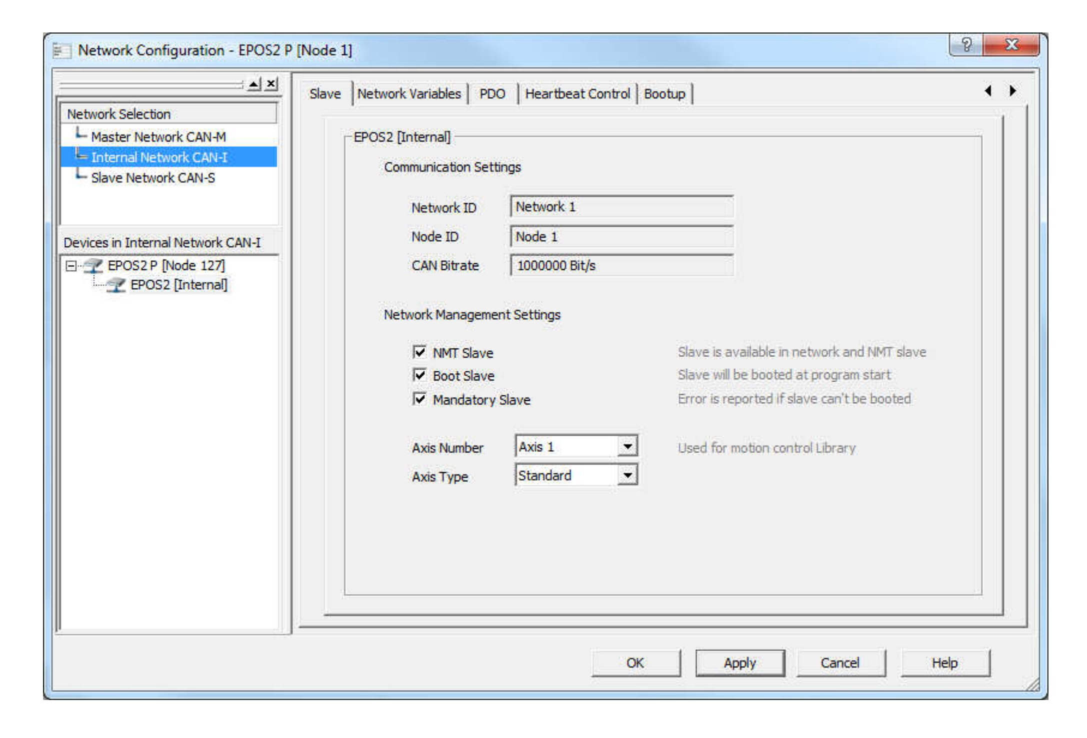


Abbildung 59: Konfiguration der EPOS2 [internal] *als Achse 1 im* Network Configuration *Tool des* EPOS Studio.

Schritt 5 Bestätige mit OK. (Und wiederum dauert das Schreiben der Konfiguration seine Zeit.)

Ein neues OpenPCS *Projekt anlegen*

Wir kehren nun zur SPS-Programmierung zurück und legen ein neues Projekt an. Dies erfolgt im *OpenPCS* Programmierwerkzeug. Das hier beschriebene Rezept folgt Kapitel 3.4 des Dokuments *Programming Reference*.

Schritt 1 Öffne im *EPOS Studio* die Tools für die *EPOS2 P*. Wähle *IEC 61131-3 Programming*.

Schritt 2 Öffne die *OpenPCS* Software durch klicken auf die *Open Programming Tool* Schaltfläche.

Schritt 3 Lege ein neues *OpenPCS* Projekt an: Wähle *New...* im *Project* Menü.

Schritt 4 Wähle im Dialogfenster den Dateityp *maxon motor ag* und das Template *EPOS2 P project*. Gib dem Projekt einen Namen (z.B. *Axis_Motion*) und definiere einen Pfad, wo es abgelegt werden soll. Bestätige mit OK.

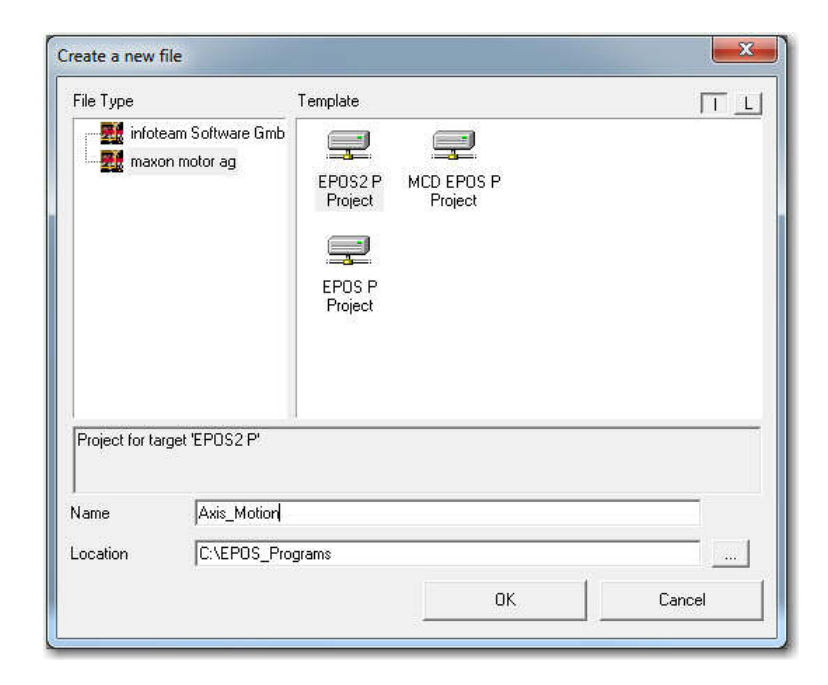


Abbildung 60: Ein SPS Projekt Axis_Motion *im* OpenPCS *anlegen für die* maxon EPOS2 P*.*

Im Projektfenster erscheint das neue Projekt mit nur gerade einer Datei: Eine leere Datei zur Deklaration von Datentypen.

Globale Variablen hinzufügen

Nun definieren wir die Achsnummer als globale Variable in einer entsprechenden Datei (vgl. auch Kapitel 3.5 des Dokuments *Programming Reference*).



Globale Variablen
Es ist nützlich, Variablen, die in mehreren Programmen Verwendung finden, als *global* zu definieren. Aber es ist kein Muss. Wir hätten die Achse auch im Deklarationsteil des Programms definieren können.

Schritt 1 Im Menü *File / New…* wähle *Declarations* und dann *Global*. Gib der Datei einen Namen, z.B. *GlobalVar*.



Abbildung 61: Eine neue Variablen-Deklarationsdatei erstellen.

Bestätige die Frage, ob die Datei zur *Active Resource* hinzugefügt werden soll, mit ja; damit wird sie in den Kompilationsprozess eingebunden. In unserem Fall ist die *Active Resource* die SPS der *EPOS2 P*. Die Datei *GlobalVar.POE* wird zum Projekt hinzugefügt und auf dem Hauptbildschirm geöffnet.

Schritt 2 Definiere eine Achsvariable, z.B. *Axis1*.
Diese Variable muss vom komplexen Typ *AXIS_REF* sein, einem vordefinierten Typ für Achsen in EPOS-Systemen. Ordne der Variable die Achsnummer 1 zu (wie in der Netzwerkkonfiguration im *EPOS Studio* definiert). Die Syntax muss so aussehen:

```
VAR_GLOBAL
  Axis1    : AXIS_REF  := (AxisNo := 1);
END_VAR
```

Schritt 3 Speichere die Datei und überprüfe die Syntax. Du findest die entsprechenden Befehle (*Save, Syntax Check*) im Menü *File* oder als Schaltfläche in der Menüleiste. (*Syntax Check* speichert die Datei zuerst). Das Ergebnis des Syntaxchecks wird unten rechts auf dem Bildschirm angezeigt. Wenn du “Glück” hast, erhältst du *0 errors, 0 warnings*. Falls nicht, vergleiche genau mit obigem Beispielcode.



Syntax Check
Lass die Syntax periodisch überprüfen. Damit vermeidest du, dass du einen Fehler in einem zu grossen Abschnitt des Programmcodes suchen musst.
In der *Structured Text* (ST) Programmiersprache müssen die einzelnen Befehle durch Semikolons getrennt werden. Zeilenumbrüche, Leerschläge und Einzüge werden ignoriert, sind aber sehr nützlich und empfohlen, um das Programm zu strukturieren und so leichter lesbar zu gestalten. Gross- und Kleinbuchstaben werden in ST nicht unterschieden und können zur weiteren Verbesserung der Lesbarkeit des Programmcodes eingesetzt werden.

Syntaxfehler
Manchmal wird eine lange Liste von Fehlern rapportiert, obwohl nur an einer Stelle etwas nicht in Ordnung ist. Gehe zum ersten Eintrag in der Liste und doppelklicke auf ihn. Der Cursor springt dann zu der Stelle im Programm, wo der Compiler den Fehler gefunden hat. Überprüfe die Zeilen vor und nach dieser Stelle. Nach Beheben dieses Fehlers und neuem Syntax Check verschwinden die anderen Einträge in der Liste meist ebenfalls (ausser es sind noch weitere Fehler aufgetreten).

8.2 Die Hauptschritte programmieren

Lernziele	Programmieren einer einfachen Achsbewegung und verwenden von Variablen und Funktionsbausteinen.

Eine neue Programmdatei erzeugen

(Vgl. auch Kapitel 3.5 des Dokuments *Programming Reference*.)

Öffne den *Create a new file* Dialog im Menü *File / New…*. Wähle *Program* als *POU-Type* und *SFC* als *IEC Language*. Gib der Programmdatei einen Namen, z.B. *MyAxisMotion*.

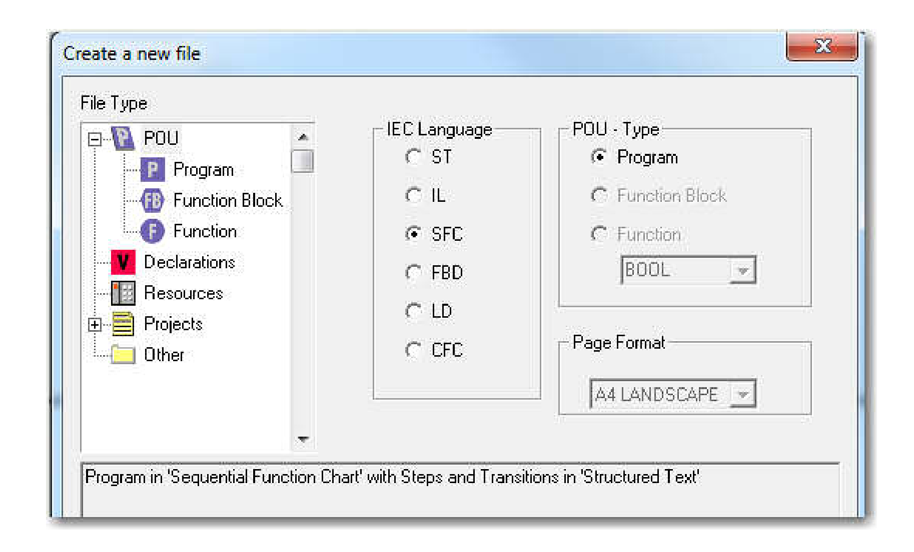


Abbildung 62: Eine neue Datei erstellen.
Hier wird eine Programmdatei erstellt in der IEC-Programmiersprache Sequential Function Chart *(SFC), wobei die Schritte und Transitionen in* Structured Text *(ST) programmiert sind.*

Bestätige die Frage, ob die Datei zur *Active Resource* hinzugefügt werden soll, mit ja. *MyAxisMotion.SFC* wird zum Projekt hinzugefügt und auf dem Hauptbildschirm geöffnet. Das SFC-Programmfenster enthält schon einen ersten Schritt, mit der Bezeichnung *Init*, und eine Transition.



Bibliotheken (*Libraries*) von Funktionsbausteinen für Motion Control mit EPOS2 P
Wie früher schon angemerkt, stehen eine Anzahl vordefinierte Funktionen und Funktionsbausteine für verschiedene Zwecke zur Verfügung. Neben den allgemeinen Bibliotheken, die im *IEC 61131-3* Standard enthalten sind, und einigen Funktionsbausteine im Zusammenhang mit CAN gibt es zwei spezielle Bibliotheken mit Funktionsbausteinen für die *EPOS2 P*.

Motion Control Bibliothek (MC_)
Die Funktionsbausteine in dieser Bibliothek beginnen mit MC_ und sind im Dokument *Programming Reference* detailiert beschrieben. Die MC_ Bibliothek ist mit dem PLCopen-Standard verträglich. Die wichtigsten Funktionsbausteine sind

- *MC_Reset* löscht Fehler und sperrt die Endstufe (*Disable*).
- *MC_Power* schaltet die Endstufe frei (*Enable*).
- *MC_MoveRelative* bewegt die Achse relativ zur Ist-Position.
- *MC_MoveAbsolute* bewegt die Achse zur absoluten Position.
- *MC_Home* führt eine Referenzfahrt aus.
- *MC_MoveVelocity* verwendet zur Drehzahlregelung.
- *MC_Stop* wird verwendet zum Stoppen.

Maxon Utility Bibliothek (MU_)
Die Funktionsbausteine in dieser Bibliothek beginnen mit MU_ und sind im Detail im Dokument *Programming Reference* beschrieben. The MU_Bibliothek enthält Funktionsbausteine, die spezifisch für die *EPOS2* Controller kreiert wurden, und ihre Verwendung im Projekt muss gesondert aktiviert werden (vgl. Abbildung 63).
Die wichtigsten MU_ Funktionsbausteine werden verwendet für

- das Setzen von Parametern der Referenzfahrt,
- das Lesen von Eingängen und das Setzen von Ausgängen, besonders bei speziellen Funktionalitäten,
- alternative Betriebsmodi wie Analoger Sollwert, *Interpolated Position Mode*, *Master Encoder Mode*.

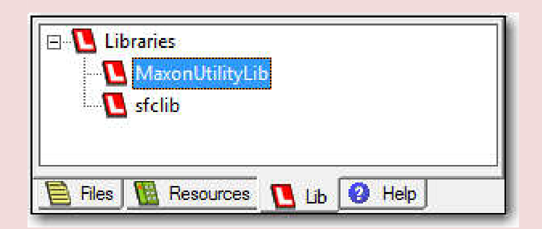


Abbildung 63: Wie man die maxon Utilities Bibliothek aktiviert. Öffne den Lib *Bildschirm und aktiviere die Verwendung von* MaxonUtilityLib *im Kontextmenü (rechte Maustaste).*



Programm-Organisations-Einheiten (*Program Organization Units*) POU

POU sind die Bausteine, mit denen ein SPS-Projekt strukturiert wird. Es gibt drei Arten von POUs: *Programs* (PROG), *Function Blocks* (FB) und *Functions* (FUN). Allen drei POUs gemeinsam ist, dass sie abgeschlossen sind und die “Kommunikation mit der äusseren Welt” wird über definierbare Schnittstellenvariablen realisiert. POUs können in Bibliotheken organisiert werden, was sie wiederverwendbar macht und eine Modularisierung der Programmierung erlaubt.

Program (PROG)

PROG ist das Hauptprogramm, wo die ganze Zuordnung zur Peripherie der SPS festgelegt wird, z.B. Ein- und Ausgänge der Hardware. Auf einer SPS-Ressource – d.h. der CPU – können verschiedene Programme gleichzeitig laufen. Jedem Programm ist ein *Task* zugeordnet, d.h. eine Bestimmung, wie und mit welcher Priorität das Programm ausgeführt werden soll. Drei Einstellungen sind möglich.

- ***cyclic***: Anweisung um Anweisung wird abgearbeitet bis zum Programmende. Dann startet das Programm neu unter Verwendung der Variablenwerte des vorangehenden Laufs. Zyklische Ausführung ist die Voreinstellung.
- ***timer***: Der *Timer*-Task startet das Programm in festen Intervallen, z.B. nach jeweils 10ms. Damit das Programm stabil läuft, muss sichergestellt sein, dass der vorangehende Durchlauf in allen Fällen rechtzeitig beendet ist.
- ***interrupt***: Interrupt-Tasks starten nur in speziellen Situationen. Zum Beispiel wenn die SPS aufgestartet oder abgeschaltet wird, wenn ein Fehler auftritt oder ein CAN Sync Befehl detektiert wird.

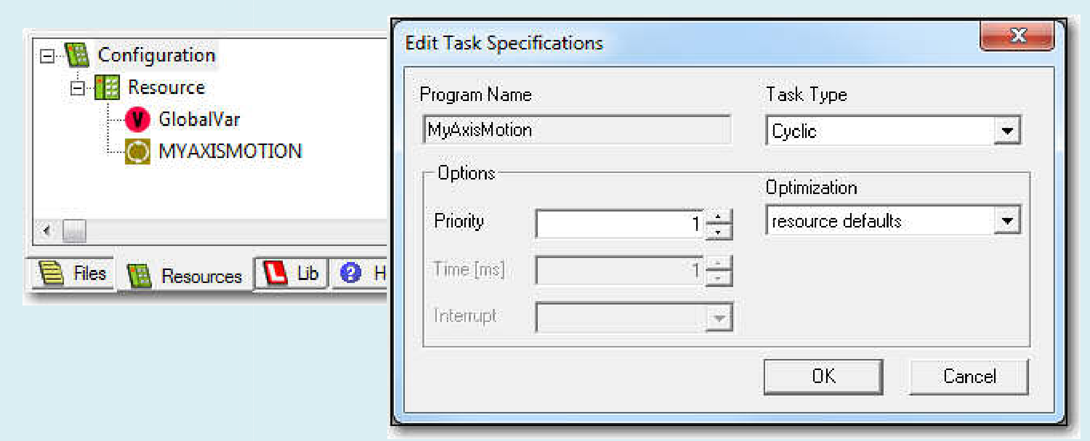


Abbildung 64: Wie der Task eines Programms eingestellt wird. Öffne den Resources *Dialog, wähle* Properties *im Kontextmenü des Programms (rechte Maustaste auf* MYAXISMOTION*).*

Bemerkung: SPS-Programme laufen zyklisch ab. Beim Start eines Durchlaufs werden die Eingangsparameter gelesen, am Ende werden die Ausgänge gesetzt. Wie lange ein Durchlauf dauert, hängt sehr stark von der Programmiersprache ab und von der Art und Weise des Programmierens. Insbesondere die Verwendung von bedingten Anweisungen (*if..then..else* Konstruktionen, *case* Befehle, Sprünge oder Schritte in SFC) kann die Menge an Programmcode, die abgearbeitet werden muss, einschränken und den Durchlauf beschleunigen.

Funktionsbausteine (Function Block, FB)
Funktionsbausteine sind wahrscheinlich die am meisten verwendeten POUs. Sie verhalten sich ähnlich wie Unterprogramme. FB arbeiten mit ihrem eignen Variablensatz (sie haben ein Gedächtnis) und müssen ähnlich einer Variable deklariert werden (Bilden einer Instanz). FB werden mit den Input-Schnittstellenvariablen aufgerufen und das Resultat des FB wird in den Output-Variablen abgelegt und kann in der aufrufenden POU verwendet werden.
FB können selber erzeugt werden oder es werden vordefinierte Funktionsbausteine aus Standard-Bibliotheken verwendet. Standard FB findet man links im *Catalog* Teil des *OpenPCS*-Bildschirms (Abbildung 50).
Ein vordefinierter Funktionsbaustein, der die Ausführung auf die **steigende Flanke** einer Input-Variablen beginnt (z.B. *Execute*) wird nur dann gestartet, wenn die Variable beim vorhergehenden Durchlauf *False* war und jetzt *True*. Aufgepasst, die SPS fährt mit der Ausführung der nächsten Anweisung fort und wartet nicht, dass der FB seine Aufgabe beendet hat. Gewisse FB brauchen ziemlich lange bis zum Ende (z.B. eine Bewegung von einigen Sekunden Dauer). Während dieser Zeit kann das SPS-Programm viele Durchläufe absolvieren. Wenn es zum zuvor gestarteten Funktionsbaustein kommt, wird dieser nicht nochmals gestartet oder abgebrochen, solange die *Execute* Input-Variable den hohen Wert behält. Darum ist es wichtig, dass die *Execute* Input-Variable des Funktionsbausteins auf dem hohen Wert bleibt, bis eine FB Output-Variable den Abschluss signalisiert (meist *Done* Output genannt). Wenn dieselbe Instanz des FB wiederverwendet werden soll, muss er zurückgesetzt, d.h. mit *Execute* auf *False* aufgerufen werden. Dies setzt dann auch den Output *Done* auf *False*. Die meisten vordefinierten Motion Control und maxon Utilities Funktionsbaustein werden mit steigender Flanke gestartet, z.B. *MC_MoveRelative* (Positionierung).
Andere vordefinierte FB reagieren auf den **Zustand** einer der Input-Variablen (oft *Enable* genannt). Sie werden in jedem Programdurchlauf ausgeführt, wenn der *Enable* Zustand hoch (TRUE) ist. Wichtige Beispiele für ein solches Verhalten sind *MC_Power* (Enable der Endstufe) und FB, die physische Eingänge lesen, z.B. *MU_GetDigitalInput*.
Die Ausführung selbstdefinierter FB hängt von der Programmierung ab.

Function (FUN)
Funktionen (FUN) haben kein Gedächtnis. Sie werden in jedem Durchlauf des SPS-Programms sofort ausgeführt und geben einen Resultatwert zurück. Funktionen werden oft für mathematische Operationen verwendet. Typische Beispiele sind trigonometrische Funktionen, Vergleiche von Variablenwerten oder Umwandlungen von Variablentypen.

Den Init *Schritt programmieren*

Wie der Name schon sagt, möchten wir diesen ersten Schritt zur Initialisierung der Achse verwenden. Die Achse muss zurückgesetzt, d.h. die Endstufe gesperrt (Disable) und allfällige Fehler gelöscht werden. Diese Initialisierung bereitet das Feld für die weitere Programmierung. Der vordefinierten Motion Control Funktionsbaustein *MC_Reset* erledigt diese Aufgabe.
Ebenfalls im *Init* Schritt möchten wir die Achse freischalten (Enable). Wiederum können wir einen vordefinierten Motion Control Funktionsbaustein verwenden: *MC_Power*.

Schritt-für-Schritt-Anleitung für den *Init* Schritt, beginnend mit der Variablendeklaration:

Schritt 1 Deklariere *Axis1* als externe Variable. Das bedeutet quasi dem *MyAxisMotion* Programm zu sagen, dass *Axis1*, die wir als globale Variable bezeichnet hatten, verwendet wird. Bezüge zu globalen Variablen müssen im Abschnitt *VAR_EXTERNAL … END_VAR* mit dem Variablennamen (*Axis1*) und dem Typ (*AXIS_REF*). deklariert werden. Weise keinen Achsnamen zu; dies ist schon bei der Deklaration der globalen Variablen erfolgt!

Schritt 2 Deklariere im Abschnitt *VAR … END_VAR* die Instanzen der Funktionsbausteine, die wir verwenden: *MC_Reset* und *MC_Power*. Eine gute Angewohnheit ist, die Instanzennamen mit *fb* zu beginnen. Damit ist klar, dass dies Funktionsbausteine sind und nicht gewöhnliche Variablen. In der nächsten Abbildung ist ein Beispiel. Vergiss nicht, die Semikolons ans Ende jeder Linie zu setzen!

Schritt 3 Klicke auf den *Init* Schritt im mittleren Teil des Programmierfensters. Im unteren Teil kannst du nun den Programmcode für diesen Schritt erfassen. Es ist ganz einfach: Rufe zuerst *fbReset* auf und sobald dieser Funktionsbaustein fertig ist, bestrome die Achse mit *fbPower*.
fbReset wird mit einer steigenden Flanke der Input-Variable *Execute* gestartet. Sinnvoll ist, *fbPower* erst dann aufzurufen, wenn *fbReset* korrekt beendet ist, was am Output *Done* angezeigt wird.

Schritt 4 Mittels einer Booleschen Variablen, z.B. *DoTransInit* bereiten wir zum Schluss die Transition zum nächsten Schritt vor. Erst wenn der Motor bestromt ist, d.h. der *Status* Output von *fbPower* auf hohem Niveau, soll das Programm mit dem nächsten Schritt weiterfahren.
Vergiss nicht, *DoTransInit* als Boolesche Variable zu deklarieren. (Wenn du willst kannst du den Startwert *false* zuweisen. Es ist aber nicht nötig, da dies sowieso der Default-Startwert ist.)

Schritt 5 Vielleicht wäre es an der Zeit einen Syntax Check aufzurufen.

```
VAR_EXTERNAL
    Axis1       : AXIS_REF;
END_VAR

VAR_GLOBAL

END_VAR

VAR
    fbPower     : MC_Power;
    fbReset     : MC_Reset;

    DoTransInit : bool := false;
END_VAR
```

Init

InitDone

Init

```
fbReset(Axis := Axis1, Execute := true);
if fbReset.done then
    fbPower(Axis := Axis1, Enable := true);
end_if;

if fbPower.status then
    DoTransInit := true;
end_if;
```

Abbildung 65: Variablendeklaration und Programmcode des Init *Schritts.*

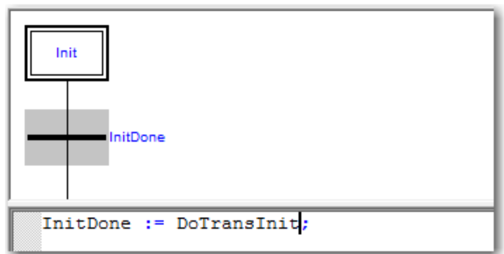


Katalog mit Funktionsbausteinen und Funktionen
Wie früher schon angemerkt, findest du die vordefinierten Funktionen (FUN) und Funktionsbausteine (FB) im *Catalog* links auf dem Bildschirm (vgl. Abbildung 56). Als Programmierer lohnt es sich, ein wenig Zeit ins Studium der vorhandenen Funktionen und Funktionsbausteine zu investieren. Dies kann dir später viel Zeit sparen. Eine Beschreibung der meisten FUN und FB im *Catalog* erhältst du durch doppelklicken darauf; dies öffnet den entsprechenden Hilfeeintrag.
Am einfachsten ist es, FUN und FB mittels *Drag-and-Drop* an die richtige Stelle in den Programmcode einzufügen. Damit erhältst du auf die Schnelle die gesamte Schnittstelle mit allen Input- und Outputparametern. Die nicht benötigten kannst du einfach löschen. Vergiss nicht die Funktionsbausteine (FB) zu deklarieren und die Instanzen geeignet zu bezeichnen. (Mit FUN ist dies nicht nötig, da sie nicht instanziert werden.)

Und hier das Rezept für die Programmierung der Transition:

Schritt 1 Gib zuerst der Transition einen neuen Namen, z.B. *InitDone*, damit das Programm leichter lesbar wird. Doppelklicke auf das Transitionssymbol und ändere den Namen. Du kannst sogar einen Kommentar hinzufügen.

Schritt 2 Formuliere die Bedingung, unter der die Transition ausgeführt werden soll. In unserem Fall soll dies geschehen, wenn *DoTransInit* eintritt.



Schritt 3 Speichere wiederum und *Check Syntax*!

Und damit hast du den ersten Schritt und die Transition programmiert.

Schritt Move1 *und Transition*

Das Ziel dieses Schrittes besteht darin, den Motor um 10’000 Quadcounts (entspricht 5 Motorumdrehungen) mit einer Drehzahl von 200 rpm weiterzudrehen. Füge eine kurze Pause nach Abschluss der Bewegung hinzu. Damit wird es einfacher, den Bewegungen des Motors zu folgen.

Schritt 1 Einen Schritt und eine Transition hinzufügen:
Markiere die Linie unterhalb der Transition *InitDone*.
Wähle aus dem Menü *Insert* den Befehl *Step/Transition*.

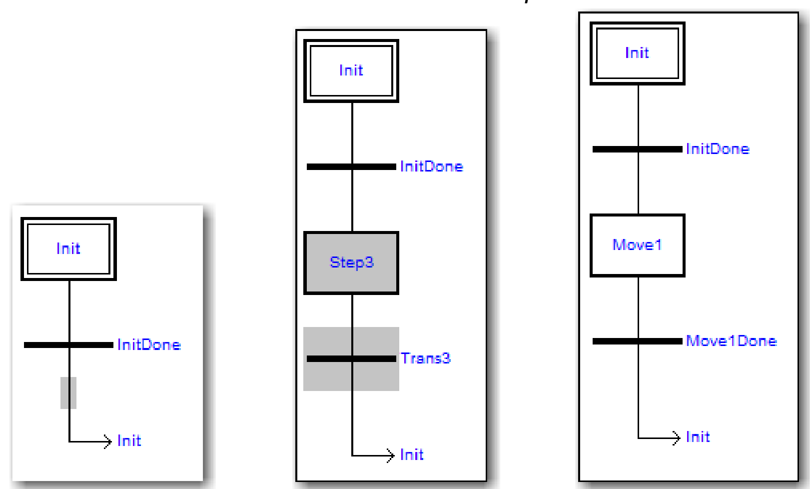


Abbildung 66: Einen Schritt und Transition in SFC hinzufügen. Markiere die Stelle, wo ein neues Element eingefügt werden soll. Wähle aus dem Insert *Menü das entsprechende Element. Durch Doppelklicken kann ein Name und Kommentar angegeben werden.*

Schritt 2 Benenne den Schritt, z.B. *Move1*, und die Transition, z.B. *Move1Done*.

Schritt 3 Zuerst schreiben wir zur Organisation etwas Programmcode in *Move1*, sozusagen um den Tisch zu bereinigen.
Falls wir je zum *Init* Schritt zurückspringen sollten, wollen wir, dass der Schritt wie beim Programmstart ausgeführt wird. Setze darum den Baustein *fbReset* des vorangehenden Schrittes zurück; d.h. rufe ihn mit *Execute* := *False* auf. Setze ebenfalls die Boolesche Variable für die Transition zurück (*DoTransInit* := *False*).

Schritt 4 *Move1* programmieren:
Füge einen Positioniervorgang hinzu. Deklariere eine Instanz des Motion Control Funktionsbausteins *MC_MoveRelative* im Variablenabschnitt. Benenne ihn beispielsweise *fbMove1*.
Füge eine kurze Pause hinzu, sobald die Bewegung abgeschlossen ist, d.h. *fbMove1.done*. Benütze einen der vordefinierten Zeit-Funktionsbausteine des *IEC 61131-3* Standards, z.B. *TON*. Taufe ihn *fbWait1* zum Beispiel. Du findest *TON* im *Catalog* links; doppelklicken gibt dir die Angaben, wie er funktioniert. Beachte das Format der Zeitvariablen, *t#3s*.

```
(* Reset the function blocks and transition flags *)
fbReset(Axis := Axis1, Execute := false);
DoTransInit := false;

(* Move and dwell *)
fbMove1(Execute := true, Axis := Axis1 , Distance := 10000,
        Velocity := 200, Acceleration := 40000 , Deceleration := 40000);

If fbMove1.done then
    fbWait1(IN := true, PT := t#3s);
end_if;

(* Set transition flag *)
DoTransMove1 := fbWait1.Q;
```

Abbildung 67: Programmcode des Schritts Move1*.*

Schritt 5 Programmiere die Transition *Move1Done.* Die Transition darf erst ausgeführt werden, wenn *fbWait1* abgelaufen ist, d.h. der Output *Q* auf den hohen Zustand geschaltet hat.

Schritt 6 Speichern und *Check Syntax*!



Limitierung der Bewegungsparameter
Wenn du Bewegungen programmierst, achte darauf, die eingestellten Grenzwerte von Drehzahl, Beschleunigung und Position jeder Achse nicht zu überschreiten. Diese Grenzen sind im Objektverzeichnis der *EPOS2 [internal]* festgelegt: *Min Position Limit, Max Position Limit, Max Profile Velocity, Max Acceleration*.

Testen

Wir können nun unser kleines Programm starten und beobachten was passiert. Gehe gemäss den Schritten in Kapitel 7 vor.

Schritt 1 Kompilieren oder *Build Active Ressource*: Klicke auf den entsprechenden Befehl im PLC Menü oder den entsprechenden Kurzbefehl. Das Resultat der Kompilation sollte ein *0 error(s) 0 warning(s)* sein.

Schritt 2 Klicke auf die *Go Online/Offline* Schaltfläche im PLC Menü oder auf den entsprechenden Kurzbefehl.

Schritt 3 Das Programm starten: Klicke auf *Coldstart*.

Schritt 4 Beobachte das Verhalten des Motors!
Zuerst ist alles wie erwartet: Die Achse wird freigeschaltet (Enable) und die Motorwelle macht 5 Umdrehungen. Nach kurzem Stillstand folgen sich *Enable* und *Disable* Vorgänge in kurzem Abstand.

Erklärung: Das Verhalten ist leicht zu verstehen. Sind die Bewegung und die Pause das erste Mal durchlaufen, springt das Programm zum *Init* Schritt zurück. Dort wird die Achse zurückgesetzt (mit *Disable*) gefolgt von einem *Enable*. Dann geht das Programm zum zweiten Mal zu Schritt *Move1*. Allerdings geschieht hier nichts weiter! Die Funktionsbaustein-Instanzen *fbMove1* und *fbWait1* werden nicht wieder gestartet, da sie nicht zurückgesetzt worden sind (*Execute* ist immer *True*). Sogar die *Move1Done* Transition ist immer *True* und das Programm springt direkt zum *Init* zurück, wo die Achse zurückgesetzt und wieder bestromt wird, und so weiter. Genau dies beobachten wir.
Beachte: Im Gegensatz zu den FBs im Schritt *Move1*, wird *fbReset* im Schritt *Init* jedes Mal ausgeführt, da er zu Beginn des Schritts *Move1* zurückgesetzt wird.

Schritt 5 Halte somit das Programm besser mit der *Stop* Schaltfläche an!

Funktionsbausteine zurücksetzen

Wenn *Move 1* immer wieder wiederholt werden soll, müssen wir die Funktionsbaustein-Instanzen *fbMove1* und *fbWait1* zurücksetzen. Am zweckmässigsten geschieht dies in einem weiteren Schritt. Somit,

Schritt 1 Füge einen Schritt und Transition nach der Transition *Move1Done* hinzu.

Schritt 2 Gib dem Schritt einen Namen, z.B. *ResetFB*, und auch der Transition, z.B. *ResetFBDone*.

Schritt 3 Programmiere den Schritt *ResetFB*:
Setze die FB des vorangehenden Schritts zurück, indem du sie mit *Execute := false* aufrufst. Setze ebenfalls die Boolesche Variable für die vorangehende Transition zurück.
Definiere eine Boolesche Variable, welche die nächste Transition steuert, z.B. *DoTransResetFB*. Die Transition soll ausgeführt werden, wenn alle Funktionsbausteine zurückgesetzt sind.

Schritt 4 Programmiere die Transition *ResetFBDone*.

Schritt 5 Modifiziere den Code, sodass nur die Bewegung wiederholt wird:
Setze den *Jump* auf den Schritt *Move1.* Doppelklick auf den Pfeil nach der letzten Transition und ändere den *Jump* auf *Move1*. Vergiss nicht, in *Move1* die Boolesche Variable für die Transition aus *ResetFB* zurückzusetzen.

Schritt 6 Speichern und *Check Syntax*.

Schritt 7 *Build Active Resource, Go Online/Offline* und *Coldstart*. Beobachte das Verhalten des Motors!

Schritt 8 Halte das Programm mit der *Stop* Schaltfläche an!

```
(* Reset the function blocks and transition flags *)
fbReset(Axis := Axis1, Execute := false);
DoTransInit := false;
DoTransResetFB := false;

(* Move and dwell *)
fbMove1(Execute := true, Axis := Axis1 , Distance := 10000,
        Velocity := 200, Acceleration := 40000 , Deceleration := 40000);

If fbMove1.done then
    fbWait1(IN := true, PT := t#3s);
end_if;

(* Set transition flag *)
DoTransMove1 := fbWait1.Q;
```

```
(* Reset the function blocks and transition flags *)
fbMove1(Execute := false, Axis := Axis1);
fbWait1(IN := false);
DoTransMove1 := false;

(* Set transition flag *)
DoTransResetFB := not fbWait1.Q and not fbMove1.done;
```

Abbildung 68: Die Codes von Schritt Move1 *(oben) und* ResetFB *(unten).*

8.3 Weitere Bewegungen programmieren

Lernziele	Programmieren einer Folge von Positionierungen und verwenden von Funktionsbausteinen zur Drehzahlregelung.

Vorwärts-rückwärts Bewegung

Wir wollen das Programm *MyAxisMotion* erweitern, sodass sich die Achse zurück zur Anfangsposition bewegt und dort während 1s stehen bleibt. Dies kann auf zwei Arten geschehen: Entweder fügen wir einen Schritt (z.B. *Move2*) und eine Transition nach *Move1* mit ähnlichem Inhalt ein. Oder wir fügen die zweite Bewegung zum Schritt *Move1* hinzu; dies ist was wir hier tun wollen.

Für beide Fälle brauchen wir zweite Instanzen von *MC_MoveRelative* und *TON*, die wir als *fbMove2* und *fbWait2* deklarieren.

Die Programmmodifikationen Schritt um Schritt:

Schritt 1 Deklariere zweite Instanzen von *MC_MoveRelative* und *TON.* Taufe diese *fbMove2* und *fbWait2*.
Schritt 2 Füge im Schritt *Move1* die zweite Bewegung hinzu (negative Distanz, andere Geschwindigkeit …), nachdem die erste Pause abgelaufen ist.
Schritt 3 Füge im Schritt *Move1* die zweite Pause hinzu, nachdem die zweite Bewegung abgeschlossen ist.
Schritt 4 Überarbeite im Schritt *Move1* die Bedingung für die Transition *Move1Done*: Pause 2 muss abgelaufen sein.
Schritt 5 Setze im Schritt *ResetFB* die zusätzlichen Funktionsbausteine des vorangehenden Schrittes zurück. Und überarbeite die Bedingung für die Transition: Alle Bausteine müsse korrekt zurückgesetzt sein.
Schritt 6 Speichern und *Check Syntax*!
Schritt 7 *Build Active Resource, Go Online/Offline* und *Coldstart*. Beobachte das Verhalten des Motors!

```
VAR
    fbReset              : MC_Reset;
    fbPower              : MC_Power;
    fbMove1, fbMove2     : MC_MoveRelative;
    fbWait1, fbWait2     : TON;

    DoTransInit          : bool := false;
    DoTransMove1         : bool := false;
    DoTransResetFB       : bool := false;
END_VAR
```

Move1

Move1Done

```
(* Reset the function blocks and transition flags *)
fbReset(Axis := Axis1, Execute := false);
DoTransInit := false;
DoTransResetFB := false;

(* Move forth and dwell *)
fbMove1(Execute := true, Axis := Axis1 , Distance := 10000,
        Velocity := 200, Acceleration := 40000 , Deceleration := 40000);
If fbMove1.done then
    fbWait1(IN := true, PT := t#3s);
end_if;

(* Move back and dwell *)
If fbWait1.Q then
    fbMove2(Execute := true, Axis := Axis1 , Distance := -10000,
            Velocity := 400, Acceleration := 40000 , Deceleration := 40000);
end_if;
If fbMove2.done then
    fbWait2(IN := true, PT := t#1s);
end_if;

(* Set transition flag *)
DoTransMove1 := fbWait2.Q;
```

Abbildung 69: Der überarbeitete Programmcode von Schritt Move1.

Bewegung bei fester Drehzahl und Stop.

Die nächste Erweiterung von *MyAxisMotion* zeigt auf, wie weitere Bewegungen über zusätzliche Schritte und Transitionen implementiert werden können. Ebenfalls führen wir zwei weitere vordefinierte Motion Control Funktionsbausteine ein: Einer um eine feste Drehzahl einzustellen (Drehzahlregelung) und einer um die Achse zu anzuhalten (Stop). Die Drehzahlregelung wollen wir nach der Hin-Her-Positionierung von *Move1* ausführen.

Hier eine mögliche Programmierung

Schritt 1 Füge einen neuen Schritt und Transition nach *Move1* ein. Benenne sie z.B. *Move2* und *Move2Done*.

Schritt 2 Definiere eine neue Variable für die Transition, z.B. *DoTransMove2*, und passe überall wo nötig das Zurücksetzen dieser Transitionvariablen an.

Schritt 3 Stelle im Schritt *Move2* eine Drehzahl an Achse 1 ein:
Deklariere und verwende eine Instanz des vordefinierten Motion Control Funktionsbausteins *MC_MoveVelocity*. Taufe ihn z.B. *fbSpeed1*.
Bemerkung: Die Drehzahl (Velocity) wird als ganzzahliger Wert ohne Vorzeichen angegeben (*unsigned double integer*). Die Drehrichtung der Bewegung wird über den vordefinierten Aufzählungstyp mit den Werten *MCpositive* oder *MCnegative* eingestellt.

Schritt 4 Stoppe die Achse nach einer gewissen Zeit in Schritt *Move2*. Deklariere und verwende eine Instanz des vordefinierten Motion Control Funktionsbausteins *MC_Stop* (genannt z.B. *fbStop*) zum Anhalten.
Bemerkung: Du könntest auch eine weitere Instanz des *MC_MoveVelocity* mit *Velocity := 0* rpm verwenden.

Schritt 5 Füge im Schritt *Move2* eine weitere Pause hinzu, nachdem die Achse zum Stopp gekommen ist.

Schritt 6 Vergiss nicht alle verwendeten Funktionsbausteine im Schritt *ResetFB* zurückzusetzen.

Schritt 7 Speichern und *Check Syntax*!

Schritt 8 *Build Active Resource*, *Go Online/Offline* und *Coldstart*. Beobachte das Verhalten des Motors!

```
VAR
     fbReset              : MC_Reset;
     fbPower              : MC_Power;
     fbMove1, fbMove2     : MC_MoveRelative;
     fbSpeed1             : MC_MoveVelocity;
     fbStop               : MC_Stop;
     fbWait1, fbWait2     : TON;
     fbWait3, fbWait4     : TON;

     DoTransInit          : bool := false;
     DoTransMove1         : bool := false;
     DoTransMove2         : bool := false;
     DoTransResetFB       : bool := false;
END_VAR
```

Move2

Move2Done

```
(* Reset the  transition flags *)
Move1Done := false;

(* Set a speed *)
fbSpeed1(Execute := true, Axis := Axis1 , Direction := MCnegative,
         Velocity := 3000, Acceleration := 60000 , Deceleration := 60000);
If fbSpeed1.InVelocity then
     fbWait3(IN := true, PT := t#5s);
end_if;

(* Stop the axis *)
If fbWait3.Q then
     fbStop(Execute := true, Axis := Axis1, Deceleration := 80000);
end_if;
If fbStop.done then
     fbWait4(IN := true, PT := t#1s);
end_if;

(* Set transition flag *)
DoTransMove2 := fbWait4.Q;
```

Abbildung 70: Wie die Programmierung von Schritt Move2 *aussehen könnte.*

9 Homing und IOs

Lernziele	Eine Einführung in nützliche Bibliotheks-Funktionsbausteine für Referenzfahrt und Handling von Ein- und Ausgängen.

In diesem Kapitel erweitern wir das *MyAxisMotion* Programm mit Funktionalität zur Ansteuerung der Ein- und Ausgänge und einer Referenzfahrt. Ziel ist es, den Stopp über einen der digitalen Eingänge auszulösen. Die Bewegung der Achse soll über einen der Ausgänge angezeigt werden; Stillstand durch einen anderen. Die Homing Methode ist *Negative Limit Switch & Index* (vgl. Kapitel 4). Homing und Handling der *EPOS2* Ein- und Ausgänge kann durch die Verwendung von Funktionsbausteinen der *maxon Utilities* (MU) Bibliothek realisiert werden.

9.1 Die Ein- und Ausgänge konfigurieren

Zuerst müssen wir die digitalen Ein- und Ausgänge der *EPOS2 [internal]* auf unsere Bedürfnisse anpassen. Wie im Kapitel 4 ausgeführt, können wir das *I/O Monitor* Tool verwenden. Das korrekte Funktionieren des *MyAxisMotion* Programms steht und fällt mit der richtigen Konfiguration.

Als Alternative könnte man diese Konfiguration auch beim Programmstart (z.B. im *Init* Schritt) durchführen, indem man all die notwendigen CANopen Objekte schreibt.



Slave Konfiguration und Programmierung
Die Slaves im Netzwerk müssen geeignet konfiguriert sein, um korrekt mit dem SPS-Programm zu funktionieren. So haben wir beispielsweise die *EPOS2 [internal]* auf den Motor- und Encodertyp eingestellt. Und wir haben gute Tuningparameter für präzise Lastbewegung gefunden. Analog können die Ein- und Ausgänge speziellen Aufgaben zugeordnet werden.

All dies muss im Objektverzeichnis des Slaves festgehalten werden. Dazu gibt es grundsätzlich zwei Möglichkeiten:

- **Konfiguration vor** dem Verbinden mit dem Netzwerk, indem man beispielsweise die richtige Konfigurationsdatei auf den Slave lädt (vgl Kapitel 6.6).
- **Konfiguration beim Programmstart** als Teil des Initialisierungsprozesses. Die richtigen Parameter werden auf alle Objektverzeichnisse geschrieben.

Die zweite Alternative hat den Vorteil, dass das korrekte Parameterset auch dann eingestellt wird, wenn die Slave Hardware ersetzt wurde. Für Anwendungen mit vielen Achsen ist dies die bevorzugte Lösung. Allerdings muss etwas mehr programmiert werden und die Initialisierung dauert länger.

In unserem Beispiel haben wir mit einer Achse eine sehr einfache Situation und wir können die I/O Konfiguration behalten, die wir am Ende von Kapitel 4 eingestellt hatten. Sie sollte wie in Abbildung 71 aussehen.

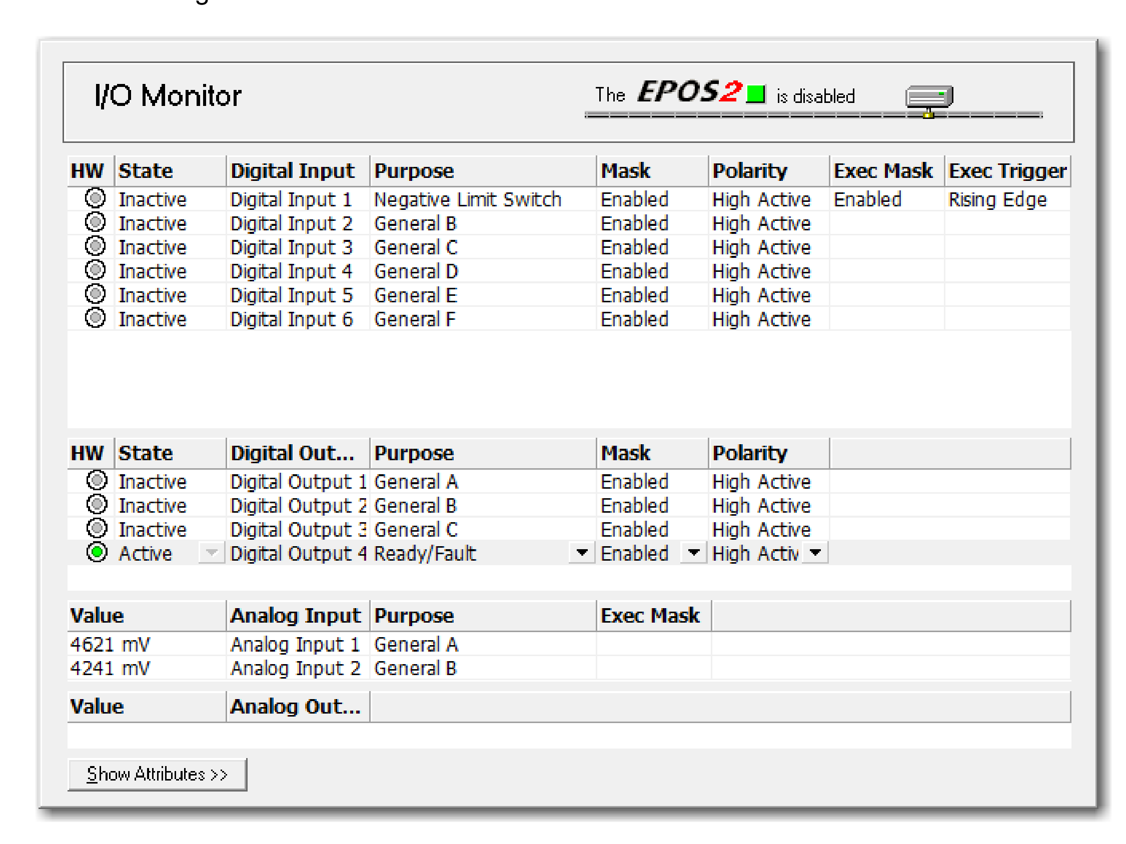


Abbildung 71: Die Konfiguration der digitalen Ein- und Ausgänge für die Programmierungen in diesem Kapitel.



***General Purpose* Ein- und Ausgänge**
Wie schon angemerkt, gehören die physischen Ein- und Ausgänge zur *EPOS2 [internal]*, d.h. zum Motion Controller. Falls sie nicht zur Regelung oder einer Funktionalität im Zusammenhang mit dieser Achse benötigt werden (z.B. als Endschalter im obigen Beispiel), können diese Ein- und Ausgänge dem Masterprogramm zur Verfügung gestellt werden. In diesem Falle müssen sie auf *General Purpose* gesetzt werden.

9.2 Referenzfahrt mit Endschalter

Da die Referenzfahrt in der Regel ja nur einmal beim Aufstarten der Maschine ausgeführt wird, ist es sinnvoll diese in den *Init* Schritt zu integrieren. Zuerst muss die Art der Referenzfahrt (*Homing Mode*) definiert werden, ausser die *EPOS2 [internal]* sei schon im Voraus mit dem richtigen *Homing Mode* konfiguriert worden. Dann kann die Referenzfahrt ausgeführt werden.

Hier ist die dazu nötige Programmierung.

Schritt 1 Deklariere Instanzen der Funktionsbausteine *MU_SetHomingParameter* und *MC_Home*, und nenne sie *fbSetHoming* und *fbHoming*.

Schritt 2 Modifiziere die Abfolge im *Init* Schritt auf: Zurücksetzen => *Homing Mode* definieren => *Enable* der Endstufe => Referenzfahrt ausführen => Transition zum nächsten Schritt. Stelle sicher, dass jede dieser Operationen abgeschlossen ist, bevor die nächste aufgerufen wird.
Tip: Drag und Drop die neuen Funktionsbausteine aus dem *Catalog* links. Doppelcklick auf die Bausteine im *Catalog* liefert Detailinformationen.

Schritt 3 Passe die Parameter von *fbSetHoming* gemäss deinen Vorstellungen an. Wichtig: Wähle *Method* 1, was eine Referenzfahrt auf den negativen Endschalter (*Negative Limit Switch & Index*) ergibt, wie du aus der Beschreibung dieses Funktionsbausteins ersiehst.

Schritt 4 Setzte die neu eingeführten Funktionsbausteine für die Referenzfahrt im Schritt *Move1* zurück.

Schritt 6 *Check Syntax* und teste das Programm. Vergiss nicht, den Endschalter auf der Printplatte während der Referenzfahrt zu aktivieren!

9.3 Ausgänge gemäss dem Achszustand setzen

Der digitale Ausgang 1 soll aktiviert sein, immer wenn sich die Achse bewegt. Der digitale Ausgang 2 soll den Stillstand der Achse anzeigen.

Für das Handling der Ausgänge benützen wir aus der MU Bibliothek den Funktionsbaustein *MU_SetAllDigitalOutput*, der alle Ausgänge setzt, jedes Mal wenn er aufgerufen wird. Gemäss Beschreibung werden beim Zurücksetzen dieses Funktionsbausteins die Zustände der digitalen Ausgänge nicht beeinflusst; sie behalten ihren digitalen Zustand (*True*, *False*) bei. Dies erlaubt es, diesen Funktionsbaustein sofort nach Verwendung zurückzusetzen, und im nächsten Programmdurchlauf ist er bereit, die Ausgänge wieder zu setzen. Die digitalen Ausgänge werden mit jedem Programmdurchlauf aktualisiert.

Der vordefinierte Funktionsbaustein *MC_ReadStatus* ermöglicht einen einfachen Zugang zum Bewegungszustand der Achse. *MC_ReadStatus* zeigt an, ob die Achse im Stillstand oder am Ausführen einer Bewegung ist (*Homing, DiscreteMotion, ContinuousMotion, Stopping* oder andere). In jedem Programmdurchlauf soll eine Instanz von *MC_ReadStatus* aufgerufen werden, um den aktuellen Bewegungszustand der Achse zu ermitteln. Wir fassen diese Information in zwei Booleschen Variablen, *InMotion* und *Standstill*, zusammen und benützen diese um die digitalen Ausgänge zu setzen.

Hier sind die dazu notwendigen Programmierschritte.

Schritt 1 Deklariere Instanzen *fbSetDigOut* und *fbStatus* der Funktionsbausteine *MU_SetAllDigitalOutputs* und *MC_ReadStatus*.
Deklariere Boolesche Variablen *InMotion* und *Standstill.*

Schritt 2 Benütze *InMotion*, um alle Zustände von *fbStatus* zusammenzufassen, die eine Bewegung signalisieren (Tipp: Schlage in der Hilfe die möglichen Output-Variablen von *MC_ReadStatus* nach). Ordne *fbStatus.Standstill* der Variable *Standstill* zu*.* Aktiviere den digitalen Ausgang 1 genau dann, wenn die Achse in Bewegung ist. Aktiviere den digitalen Ausgang 2 genau dann, wenn die Achse im Stillstand ist.
Tipp: Setze *fbSetDigOut* sofort nach Aufruf zurück. Damit werden die Ausgänge in jedem Programmdurchlauf aktualisiert. (Abbildung 72 zeigt, wie dies für den *Init* Schritt aussehen könnte.)

Schritt 3 *Check Syntax* und kopiere dieses Programmstück in alle Schritte, die eine Bewegung enthalten (*Init*, *Move1* und *Move2*). Damit ist sichergestellt, dass die Ausgänge immer korrekt gesetzt sind.

Schritt 4 *Check Syntax* nochmals und teste das Programm. Beobachte auf dem Print, wie die Ausgänge (LEDs) ändern, wenn die Achse in Bewegung ist oder stillsteht.

```
(* Setting the outputs according to the Axis state *)
fbStatus(Axis := Axis1, Enable := true);
Standstill := fbStatus.standstill;
InMotion:= fbStatus.DiscreteMotion or fbStatus.Homing
            or fbStatus.ContinuousMotion or fbStatus.Stopping;
fbSetDigOut (Axis := Axis1, Execute := true,
            GenPurpA := InMotion, GenPurpB := Standstill);
fbSetDigOut (Axis := Axis1, Execute := false);  (* Reset of fbSetDigOut *)


(* Reset the axis, and Homing *)
fbReset(Axis := Axis1, Execute := true);
if fbReset.done then
    fbSetHoming(Axis := Axis1, Execute := true, Method := 1, Offset := 0,
                Speedswitch := 100, Speedindex := 20, Acceleration := 500);
end_if;

if fbSetHoming.done then
    fbPower(Axis := Axis1, Enable := true);
end_if;

if fbPower.status then
    fbHoming(Axis := Axis1, Execute := true, Position := 0);
end_if;

(* Wait for DigIn2 to be activated *)
if fbHoming.done then
    fbReadDigIn2 (Axis := Axis1, Enable := true, Purpose := 14);
end_if;

(* Set transition flag *)
DoTransInit := fbReadDigIn2.state;
```

Abbildung 72: Der Init *Schritt*
mit dem Homing, *dem Setzen der Ausgänge und Lesen des digitalen Eingang 2.*

9.4 Eingänge lesen

Wir erweitern unser Programm, sodass der Ablauf durch die physischen Eingänge gesteuert wird.

Nach der Initialisierung soll das Programm warten, bis ein digitales Signal am Eingang 2 den weiteren Ablauf freischaltet. Auf ähnliche Weise soll die Phase der Drehzahlregelung in Schritt *Move2* über eine ansteigende Flanke beim Eingang 2 gestoppt werden und nicht mehr über die Zeitdauer.

Hier sind die dazu notwendigen Programmierschritte (vgl. auch mit Abbildung 72).

Schritt 1 Deklariere *fbReadDigIn2* als Instanz des Funktionsbausteins *MU_GetDigitalInput*.

Schritt 2 Modifiziere den Ablauf im *Init* Schritt zu: Zurücksetzen => *Homing Mode* festlegen => *Enable* der Endstufe => *Homing* ausführen => digitalen Eingang 2 lesen => Transition zum nächsten Schritt.

Schritt 3 Definiere die *Purpose* Input-Variable von *fbReadDigIn2* sodass der physische Eingang 2 gelesen wird. Gemäss unseren Einstellungen in Abbildung 71 ist dieser Eingang *General Purpose B*, was bedeutet, dass die *Purpose* Input-Variable von *fbReadDigIn2* auf den Wert 14 eingestellt werden muss.

Schritt 4 Benütze die Output-Variable *fbReadDigIn2.state*, um die Transition von *Init* nach *Move1* zu steuern.

Schritt 5 Setze *fbReadDigIn2* im Schritt *Move1* zurück.

Schritt 6 Benütze *fbReadDigIn2* ebenfalls, um die Drehzahl im Schritt *Move2* zu stoppen. Eigentlich geht es darum, *fbWait3* zu ersetzen.
Steuere die Transition zum nächsten Schritt mit *fbReadDigIn2.state*.

Schritt 7 Vergiss nicht, *fbReadDigIn2* im Schritt *ResetFB* zurückzusetzen.

Schritt 8 *Check Syntax* und teste das Programm.

10 Selbst-definierte Funktionsbausteine

Lernziele	Erzeugen und anwenden eines selbst-definierten Funktionsbausteins. Programmierung in Funktionsbausteinsprache (*Function Block Diagram,* FBD) Sprache.

Die Code-Sequenz, um den Achsstatus zu lesen und die digitalen Ausgänge zu setzen (vgl. Kapitel 9.3), wird an diversen Stellen im Programm verwendet; aktuell in jedem Schritt, der Bewegung enthält. Statt die Sequenz mehrmals zu kopieren (mit der Gefahr, sie an vielen Stellen ändern zu müssen), könnten wir die ganze Aktion in einen selbstdefinierten Funktionsbaustein stecken, den wir bei Bedarf aufrufen.

Zur Übung programmieren wir den Funktionsbaustein nicht in *Structured Text*, sondern in der graphischen Funktionsbaustein-Sprache (*FBD*, *Function Block Diagram)*.

10.1 Einen Funktionsbaustein erzeugen

Öffne den *Create a new file* Dialog im Menü *File / New.* Wähle im Abschnitt rechts *Function Block* als *POU-Type* und *FBD* als *IEC Language*. Gib dem Baustein einen Namen, z.B. *ShowAxisState*.

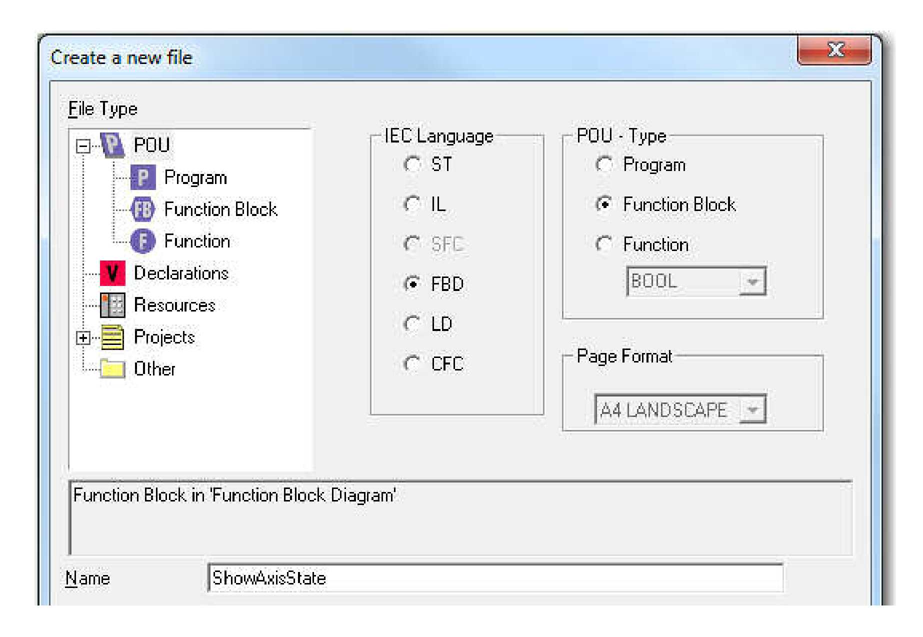


Abbildung 73: Eine neue Datei erstellen in OpenPCS*.*
Hier wird ein Funktionsbaustein erzeugt in der IEC-Sprache Function Block Diagram (FBD).

10.2 In Function Block Diagram (FBD) programmieren

Im Folgenden die Schritte zur Programmierung von *ShowAxisState.* Vieles davon handelt von der Bedienung des FBD-Editors.

Schritt 1 Variablendeklaration des Funktionsbausteins:
Wie gewohnt wird die Achsnummer als Input/Output-Variable deklariert. Definiere eine Input-Variable *Enable*. Der Aufruf von *ShowAxisState* mit *Enable* als *True* soll ein Lesen des Achsstatus auslösen und dann das Setzen der Ausgänge. Weiter brauchen wir Instanzen der entsprechenden Funktionsbausteine aus den Bibliotheken.
Aus Gründen der Vollständigkeit wollen wir ebenfalls die üblichen Output-Variablen *Done*, *Error*, und *ErrorID* definieren, auch wenn wir sie nicht benützen.
Speichern und *Check Syntax*!

```
VAR_IN_OUT
    Axis            : AXIS_REF;
END_VAR

VAR_INPUT
    Enable          : bool;
END_VAR

VAR_OUTPUT
    Done            : bool; (* indicates that the FB is ready again*)
    Error           : bool; (* no error managment set up yet *)
    ErrorID         : DINT;
END_VAR

VAR
    fbSetDigOut     : MU_SetAllDigitalOutputs;
    fbStatus        : MC_ReadStatus;
END_VAR
```

Abbildung 74: Variablendeklaration von ShowAxisState.

Schritt 2 Füge einen *MC_ReadStatus* Funktionsbaustein im Programmierfenster ein. Wähle dazu den entsprechenden Befehl aus dem *Insert* Menü oder aus dem Kontextmenü (rechte Maustaste auf dem Programmierfenster) oder benütze die Schaltfläche mit dem Kurzbefehl.

Schritt 3 Benenne *MC_ReadStatus* korrekt, gemäss dem oben deklarierten Namen, hier *fbStatus*.

Schritt 4 Ordne die Input-Variablen *Axis* und *Enable* zwei der Felder auf dem rechten Rand zu. Benütze die rechte Maustaste auf dem spezifischen Feld und *Insert Variable….*

Schritt 5 Verbinde diese Input-Variablen mit *fbStatus*.
Aktiviere die beiden Endpunkte der Verbindung und *Insert Connection.* Benütze das *Insert* Menü oder die Schaltfläche mit dem Kurzbefehl oben oder die Tastenkombination *ctrl B*.

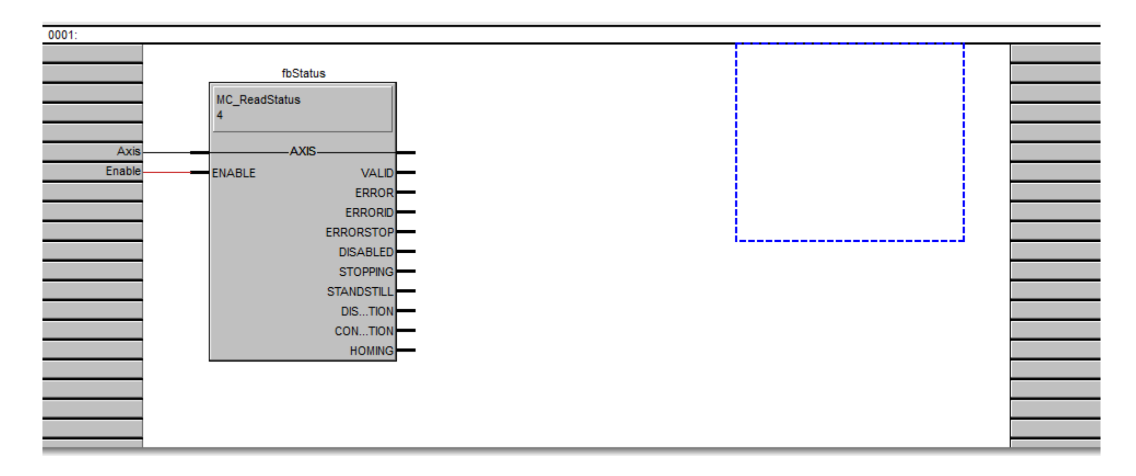


*Abbildung 75: Das Programmierfenster nach Schritt 5 (*FBD*).*

Schritt 6 Füge eine Instanz *fbSetDigOut* von *MU_SetAllDigitalOutputs* analog zu Schritt 2 und 3 ein.
Verbinde den *Axis*-Output von *fbStatus* mit dem *Axis*-Input von *fbSetDigOut*. Mit dieser Verbindung wird *fbSetDigOut* für die korrekte Achse aufgerufen.
Verbinde den *Valid*-Output von *fbStatus* mit dem *Execute*-Input von *fbSetDigOut*. Damit wird *fbSetDigOut* nur aufgerufen, wenn der Achsstatus korrekt gelesen wurde. Da *fbSetDigOut* nur mit einer steigenden Signalflanke am *Execute*-Input funktioniert, müssen wir diesen Funktionsbaustein nach jedem Aufruf zurücksetzen (vgl. Schritt 9).

Schritt 7 Verbinde den *Standstill*-Output von *fbStatus* mit dem *GenPurpB*-Input von *fbSetDigOut*. Damit wird wie gewünscht der digitale Ausgang 2 immer dann gesetzt, wenn die Achse im Stillstand ist.

Schritt 8 Immer wenn die Achse in einem der Zustände *Stopping, Discrete Motion, Continuous Motion* oder *Homing* ist, soll der digitale Ausgang 1 aktiviert sein. Füge dazu die vordefinierte Funktion *OR_BOOL_FBD* ein, die einem logischen ODER entspricht.
Verbinde die Outputs von *fbStatus*, die eine Bewegung anzeigen, mit den Inputs der ODER Funktion. Du kannst zusätzliche Inputs in *OR_BOOL_FBD* mit *Append Input* Befehl erzeugen (rechte Maustaste auf die Funktion).
Verbinde den Output von *OR_BOOL_FBD* mit dem *GenPurpA*-Input von *fbSetDigOut*.
Ich denke, es wäre Zeit für einen *Check Syntax*!

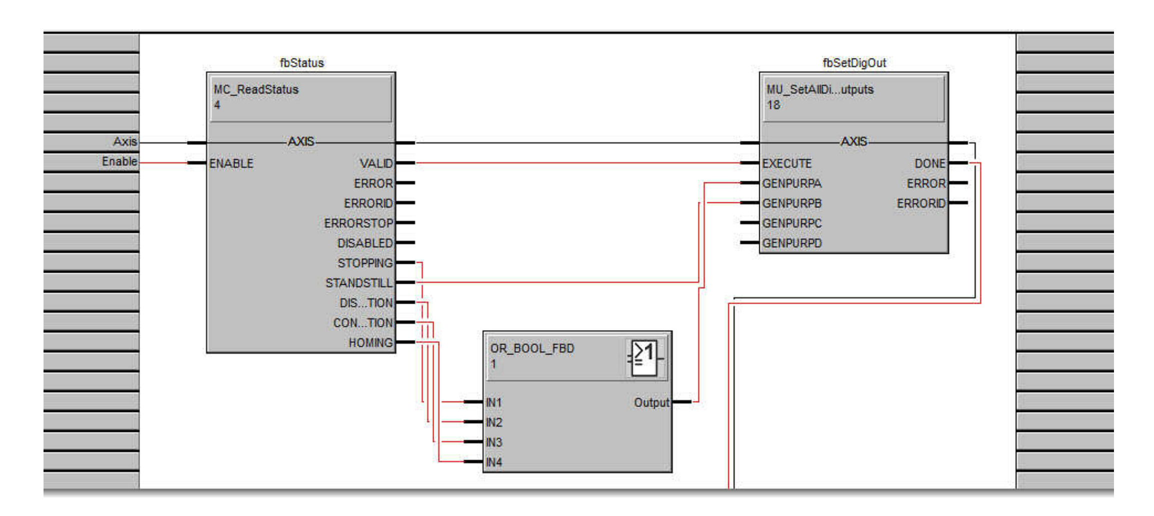


*Abbildung 76: Das Programmierfenster nach Schritt 8 (*FBD*).*

Schritt 9 Sobald die Ausgänge richtig gesetzt sind (d.h. *fbSetDigOut.done* ist *True*), muss *fbSetDigOut* zurückgesetzt, d.h. mit *Execute = False* aufgerufen werden.
Füge eine Kopie von *fbSetDigOut* ein und verbinde *fbSetDigOut.done* mit *Execute* der Kopie und *Negate Input* (rechte Maustaste). Vergiss nicht die Achsnummer ebenfalls zu verbinden. Speichern und *Check Syntax*!

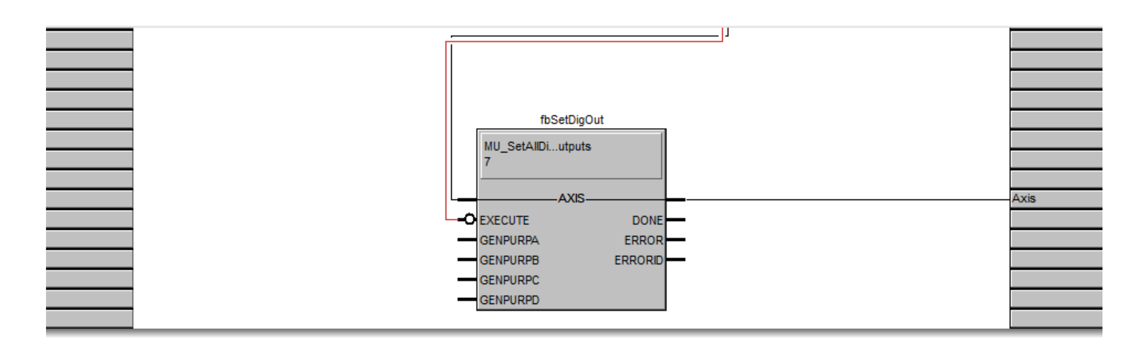


Abbildung 77: Zurücksetzen programmieren (unterer Teil von Abbildung 76).

10.3 Selbstdefinierte Funktionsbausteine benützen

Jetzt gilt es, das Hauptprogramm mit folgenden Schritten anzupassen.

Schritt 1 Deklariere *fbShowAxisState* als Instanz des neu definierten Funktionsbausteins *ShowAxisState* in der Variablendeklaration des Hauptprogramms.

Schritt 2 Lösche alle Variablen und Funktionsbaustein-Instanzen, die nicht mehr nötig sind: *fbSetDigOut, fbStatus, InMotion, Standstill*.

Schritt 3 Ersetze in allen Schritten mit Bewegung den Teil des Programmcodes, der die Ausgänge gemäss dem Achszustand setzt, mit dem Aufruf von *fbShowAxisState*. Der *Init* Schritt sieht nun so wie in Abbildung 78 aus.

Schritt 4 Speichern und *Check Syntax*! Kompilieren, *Download* und überprüfen, dass alles richtig läuft.

```
(* Setting the outputs according to the Axis state *)
fbShowAxisState (Axis := Axis1, Enable := true);

(* Reset the axis, and Homing *)
fbReset(Axis := Axis1, Execute := true);
if fbReset.done then
    fbSetHoming(Axis := Axis1, Execute := true, Method := 1, Offset := 0,
                Speedswitch := 100, Speedindex := 20, Acceleration := 500);
end_if;

if fbSetHoming.done then
    fbPower(Axis := Axis1, Enable := true);
end_if;

if fbPower.status then
    fbHoming(Axis := Axis1, Execute := true, Position := 0);
end_if;

(* Wait for DigIn2 to be activated *)
```

Abbildung 78: Die Ausgänge zur Anzeige des Achszustands einstellen mit den Funktionsbausteinen im Init *Schritt. (Verglichen mit Abbildung 72 sind nur die ersten Programmzeilen gezeigt).*

Teil 4: Anhang, Verweise und Index

11 Anhang

11.1 Motor- und Encoder-Datenblätter

Die maxon Motor-Encoder-Kombination mit Bestellnummer 301782 besteht aus

- *EC-max 30* Motor mit Bestellnummer **272763**
- *MR Encoder* mit Bestellnummer **225778**

Motordatenblätter

EC-max 30 Ø30 mm, bürstenlos, 60 Watt

Lagerprogramm
Standardprogramm
Sonderprogramm (auf Anfrage)

		Artikelnummern			
		272762	272763	272764	272765
Motordaten					
Werte bei Nennspannung					
1 Nennspannung	V	12	24	36	48
2 Leerlaufdrehzahl	min^{-1}	7980	9340	9490	9350
3 Leerlaufstrom	mA	302	191	130	95.4
4 Nenndrehzahl	min^{-1}	6590	8040	8270	8130
5 Nennmoment (max. Dauerdrehmoment)	mNm	63.6	60.7	63.7	64.1
6 Nennstrom (max. Dauerbelastungsstrom)	A	4.72	2.66	1.88	1.4
7 Anhaltemoment	mNm	381	458	522	519
8 Anlaufstrom	A	26.8	18.8	14.5	10.7
9 Max. Wirkungsgrad	%	80	81	82	82
Kenndaten					
10 Anschlusswiderstand Phase-Phase	Ω	0.447	1.27	2.48	4.49
11 Anschlussinduktivität Phase-Phase	mH	0.049	0.143	0.312	0.573
12 Drehmomentkonstante	mNm/A	14.2	24.3	35.9	48.6
13 Drehzahlkonstante	min^{-1}/V	672	393	266	197
14 Kennliniensteigung	min^{-1}/mNm	21.2	20.6	18.4	18.2
15 Mechanische Anlaufzeitkonstante	ms	4.86	4.73	4.21	4.17
16 Rotorträgheitsmoment	gcm^2	21.9	21.9	21.9	21.9

Abbildung 79: Motordatenblatt EC-max 30.

Spezifikationen

	Thermische Daten	
17	Therm. Widerstand Gehäuse-Luft	7.4 K/W
18	Therm. Widerstand Wicklung-Gehäuse	0.5 K/W
19	Therm. Zeitkonstante der Wicklung	2.76 s
20	Therm. Zeitkonstante des Motors	1000 s
21	Umgebungstemperatur	-40…+100°C
22	Max. Wicklungstemperatur	+155°C

	Mechanische Daten (vorgespannte Kugellager)		
23	Grenzdrehzahl		15000 min^{-1}
24	Axialspiel bei Axiallast	< 6.0 N	0 mm
		> 6.0 N	0.14 mm
25	Radialspiel		vorgespannt
26	Max. axiale Belastung (dynamisch)		5 N
27	Max. axiale Aufpresskraft (statisch)		98 N
	(statisch, Welle abgestützt)		1300 N
28	Max. radiale Belastung, 5 mm ab Flansch		25 N

	Weitere Spezifikationen	
29	Polpaarzahl	1
30	Anzahl Phasen	3
31	Motorgewicht	305 g

Motordaten gemäss Tabelle sind Nenndaten.

Anschlüsse Motor (Kabel AWG 20)

rot	Motorwicklung 1	Pin 1
schwarz	Motorwicklung 2	Pin 2
weiss	Motorwicklung 3	Pin 3
	N.C.	Pin 4
Stecker	**Artikelnummer**	
Molex	39-01-2040	

Anschlüsse Sensoren (Kabel AWG 26)

gelb	Hall-Sensor 1	Pin 1
braun	Hall-Sensor 2	Pin 2
grau	Hall-Sensor 3	Pin 3
blau	GND	Pin 4
grün	V_{Hall} 3…24 VDC	Pin 5
	N.C.	Pin 6
Stecker	**Artikelnummer**	
Molex	430-25-0600	

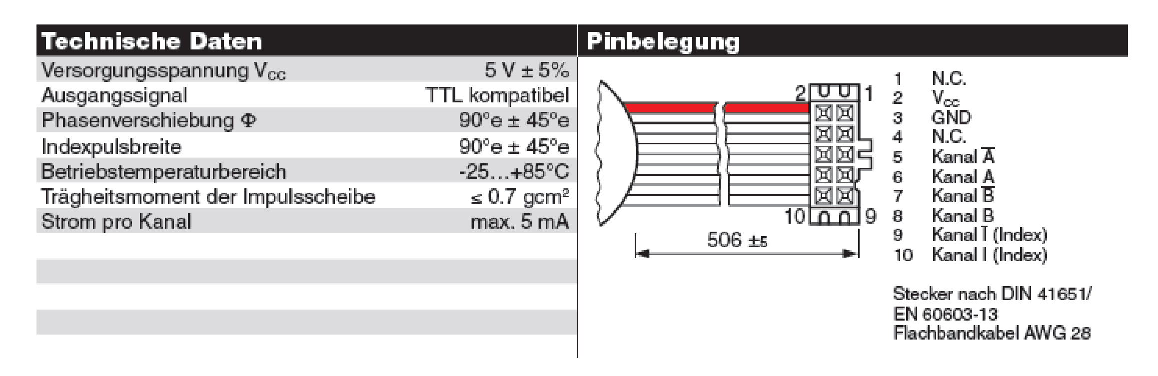
Schaltbild für Hall-Sensoren siehe S. 35

Abbildung 80: Auszug aus dem Motordatenblatt EC-max 30*: Spezifikationen.*

Encoderdatenblatt

Encoder MR Typ ML, 128–1000 Impulse, 3 Kanal, mit Line Driver

Lagerprogramm / Standardprogramm / Sonderprogramm (auf Anfrage)	Artikelnummern				
	225771	225773	225778	225805	225780
Typ					
Impulszahl pro Umdrehung	128	256	500	512	1000
Anzahl Kanäle	3	3	3	3	3
Max. Impulsfrequenz (kHz)	80	160	200	320	200
Max. Drehzahl (min^{-1})	37500	37500	24000	37500	12000



Technische Daten	
Versorgungsspannung V_{CC}	5 V ± 5%
Ausgangssignal	TTL kompatibel
Phasenverschiebung Φ	90°e ± 45°e
Indexpulsbreite	90°e ± 45°e
Betriebstemperaturbereich	-25…+85°C
Trägheitsmoment der Impulsscheibe	≤ 0.7 gcm²
Strom pro Kanal	max. 5 mA

Abbildung 81: Auszüge aus dem MR Encoder *Datenblatt.*

12 Verweise, Glossar

12.1 Liste der Abbildungen

Abbildung 1: Der Inhalt des EPOS2 P Starter Kit. 7
Abbildung 2: Wie das EPOS2 P Starter Kit verkabelt wird. 10
Abbildung 3: Schematische Übersicht der EPOS2 P........ 11
Abbildung 4: Die Dateistruktur der maxon EPOS Softwareinstallation. 12
Abbildung 5: Die Dokumentstruktur der maxon EPOS2 P........ 13
Abbildung 6: Wo sich das New Project Symbol befindet. 14
Abbildung 7: New Project Wizard, Schritt 1. 15
Abbildung 8: New Project Wizard, Schritt 2. 15
Abbildung 9: Der EPOS Studio Bildschirm mit dem geöffneten Workspace Tab. 16
Abbildung 10: Liste der Fehler und Warnungen im Status Fenster. 16
Abbildung 11: Der Communication Tab auf dem EPOS Studio Bildschirm. 17
Abbildung 12: Das richtige Gerät auswählen im Wizards Tab........ 18
Abbildung 13: Startup Wizard, Schritt 1. 19
Abbildung 14: Startup Wizard, Schritt 2. 20
Abbildung 15: Startup Wizard, Schritt 4. 20
Abbildung 16: Startup Wizard, Schritt 5. 21
Abbildung 17: Startup Wizard, Schritt 6. 22
Abbildung 18: Startup Wizard, Schritt 7. 23
Abbildung 19: Die Signale eines Inkrementalencoders. 26
Abbildung 20: Regulation Tuning Wizard: der Auto Tuning Bildschirm. 28
Abbildung 21: Regulation Tuning Wizard........ 30
Abbildung 22: PID Controller........ 32
Abbildung 23: Schematische Darstellung der Vorsteuerung. 34
Abbildung 24: Master-Slave-Architektur mit EPOS2 Slaves........ 35
Abbildung 25: Wie man den Profile Position Mode öffnet........ 36
Abbildung 26: Geschwindigkeitsprofile für eine Positionierung. 37
Abbildung 27: Darstellung einer Positionsreglung 39
Abbildung 28: Wo man die Application Notes Collection findet. 40
Abbildung 29: Target reached. 42
Abbildung 30: Prinzip des Homing mit Indexkanal des Encoders........ 45
Abbildung 31: Geschwindigkeitssignale eines Motors mit MR Encoder. 49
Abbildung 32: Geschwindigkeitssignale eines Motors mit optischem Encoder. 50
Abbildung 33: Schematische Darstellung des Master Encoder Mode. 59
Abbildung 34: Schematsche Darstellung des Step Direction Mode........ 60
Abbildung 35: Physikalischer Aufbau des CANbus........ 62
Abbildung 36: CANopen Geräteprofil 63
Abbildung 37: Zugriff auf das Objektverzeichnis (Object Dictionary) 64
Abbildung 38: Die verschiedenen Object Dictionaries im EPOS2 P Netzwerk........ 65
Abbildung 39: Einige Standard CANopen Geräteprofile (Quelle CiA Website). 66
Abbildung 40: Objektverzeichnis des Motion Controller EPOS2. 67
Abbildung 41: Wo man die Refresh Rate ändert........ 68
Abbildung 42: Das IEC-61131 Programming Tool im EPOS Studio öffnen........ 74
Abbildung 43: Öffnen des OpenPCS Tool mit einem bestehenden Sample Project. 75

Abbildung 44: Der Bildschirm des SimpleMotionSequence Projekts. 76
Abbildung 45: Build Active Ressource 76
Abbildung 46: Errors und Warnings des Kompiliervorgangs 77
Abbildung 47: Verbindung zur SPS. Go Online/Offline. 77
Abbildung 48: Coldstart des SPS Programms 77
Abbildung 49: Kurzbefehle im OpenPCS. 78
Abbildung 50: Das Fenster mit den Projektdateien. 79
Abbildung 51: Die State Machine des SimpleMotionSequence Projekts 80
Abbildung 52: Die PROG_Main.SFC Datei von SimpleMotionSequence. 81
Abbildung 53: Programmcode (ohne Kommentare) des Init Schritts 82
Abbildung 54: fbInit.. 83
Abbildung 55: Programmcode von FB_MAIN_Init.FBD. 84
Abbildung 56: Catalog der verfügbaren Funktionsbausteine (und Funktionen). 85
Abbildung 57: Die Variablendeklaration des Funktionsbausteins FB_Main_Init.FBD. 86
Abbildung 58: Mögliche Abfolge in AxisMotion 89
Abbildung 59: Konfiguration der EPOS2 [internal] als Achse 1 90
Abbildung 60: Ein SPS Projekt Axis_Motion im OpenPCS anlegen 91
Abbildung 61: Eine neue Variablen-Deklarationsdatei erstellen. 92
Abbildung 62: Eine neue Datei erstellen. 94
Abbildung 63: Wie man die maxon Utilities Bibliothek aktiviert. 95
Abbildung 64: Wie der Task eines Programms eingestellt wird. 96
Abbildung 65: Variablendeklaration und Programmcode des Init Schritts. 99
Abbildung 66: Einen Schritt und Transition in SFC hinzufügen. 102
Abbildung 67: Programmcode des Schritts Move1. 103
Abbildung 68: Die Codes von Schritt Move1 (oben) und ResetFB (unten). 105
Abbildung 69: Der überarbeitete Programmcode von Schritt Move1. 107
Abbildung 70: Wie die Programmierung von Schritt Move2 aussehen könnte. 109
Abbildung 71: Die Konfiguration der digitalen Ein- und Ausgänge 111
Abbildung 72: Der Init Schritt 114
Abbildung 73: Eine neue Datei erstellen in OpenPCS. 116
Abbildung 74: Variablendeklaration von ShowAxisState. 117
Abbildung 75: Das Programmierfenster nach Schritt 5 (FBD). 118
Abbildung 76: Das Programmierfenster nach Schritt 8 (FBD). 119
Abbildung 77: Zurücksetzen programmieren (unterer Teil von Abbildung 76). 119
Abbildung 78: Die Ausgänge zur Anzeige des Achszustands einstellen 120
Abbildung 79: Motordatenblatt EC-max 30. 121
Abbildung 80: Auszug aus dem Motordatenblatt EC-max 30: Spezifikationen. 122
Abbildung 81: Auszüge aus dem MR Encoder Datenblatt. 123

12.2 Liste der Kästchen



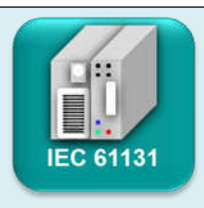
EPOS Info

- Das neueste EPOS Studio 9
- Betriebsanleitungen und Software-Dokumentation 12
- Projekt im EPOS Studio 14
- Fehler (Errors) und Warnungen (Warnings) 16
- Motor- und Encoder 25
- Expert Tuning und manuelles Tuning 29
- Grüne und rote LED 37
- Einschränkungen der Variablentypen bei EPOS-Systemen 65
- Objektverzeichnisse der EPOS2 *P* 65
- Firmware Specification 65
- Parameter speichern 69
- PDO und SDO in EPOS-Systemen 72
- Blaue LED: Anzeige des SPS-Programmstatus 75
- EPOS Datentyps 87
- Programming Reference 88
- General Purpose Ein- und Ausgänge 111



Hintergrund zu Motion Control

- Kommutierung von Motoren mit und ohne Bürsten 24
- Positions- und Drehzahlbestimmung mit Inkrementalencoder 26
- Encoderimpulszahl und Regelungsdynamik 27
- Feedback und Feed-Forward 32
- Master, **Slaves und on-line Kommandierung** 35
- Enable und Disable der Endstufe 38
- Die Motion Control **Regelkreise** 38
- Positioniergenauigkeit 42Maximaler **Schleppfehler (*Maximum Following Error*)** 43
- Was bedeutet Referenzfahrt (Homing) und warum wird sie gebraucht? 45
- Positionieren ohne Profil 47
- Die Geschwindigkeitssignale im Data Recorder 49
- Ungewolltes Hochlaufen 52
- Endschalter (Limit Switch) und Referenzschalter (Home Switch**)** 55
- Device Enable und Quick Stop 55
- Position Marker 55
- Ready/Fault 56
- Position Compare 56
- Haltebremse (Holding Brake**)** 56



IEC

- SPS Hintergrundinformation 70
- Variable 86
- Programm-Organisations-Einheiten (Program Organization Units) POU 96



OpenPCS Info

- Das PLC Menü und die Kurzbefehle im OpenPCS 78
- Farbcodierung des Programmtextes im ***OpenPCS*** 85



Best Practice

- Arbeitsverzeichnis 8
- Den USB-Treiber finden 10
- Diagramm-Zoom 31
- Data Recorder 41
- Einschränkende Bewegungsparameter und Dämpfung 46
- Deinen eigenen Objektfilter erzeugen 69
- Refreshing rate des Objektverzeichnisses 68
- Parameter speichern 54
- Hardware Enable 57
- Limitierung und Dämpfung der Systemreaktion 58
- Globale Variablen 92
- Syntax Check 93
- Syntaxfehler 93
- Katalog mit Funktionsbausteinen und Funktionen 100
- Limitierung der Bewegungsparameter 103
- Slave Konfiguration und Programmierung 110

12.3 Literatur

- Feinmess: www.feinmess.de/mt_all/glossar.htm
- John: K.-H. John, M. Tiegelkamp “SPS-Programmierung mit IEC 61131-3”, 3rd Edition, Springer Verlag 2000. ISBN 3-540-66445-9
- Wikipedia: www.wikipedia.org
- CAN in Automation: www.can-cia.org

12.4 Index

A

Accuracy
  position accuracy, 42
Analog position control, 60
Analog set value, 95
Analoge Drehzahlregelung, 57
Analoge Positionierung, 58
Analoger Sollwert, 57
Ausgang
  Digitale Ausgänge, 53
  General Purpose, 111, 126
AXIS_REF, 87

B

Beschleunigung
  Einheit, 27
  maximale, 46
Betriebsanleitung, 12, 126
  Application Notes Collection, 40
  Firmware Specification, 12
  Programming Reference, 12
Bibliothek
  MU maxon utility, 95
Bibliothek
  MC motion control, 95
Bremse
  Haltebremse, 56
Bremsen
  Einheit, 27

C

CAN, 62
  CANopen, 62
  CANopen Gerät, 63
  CANopen Geräteprofil, 66
  CiA, CAN in Automation, 61
  Kommunikation, 71
  Prozess-Daten-Objekt PDO, 71
  Service-Daten-Objekt SDO, 72
Current
  Nominal current, 22

D

Data Recording, 40
Device Configuration File. see Electronic data sheet
Disabled, 38
Dokumentation, 12
Drehzahl
  Grenzdrehzahl, 22
Drehzahlregelung
  Profile Velocity Mode, 48
  Velocity Mode, 51

E

Echtzeit, 68, 71
Eingang
  Analoge Eingänge, 54
  Digitale Eingänge, 54
  General Purpose, 111, 126
Eingang/Ausgang
  I/O Monitor, 53
Electronic data sheet, 70
Enable
  Device Enable, 55, 126
  Enabled, 38
  Hardware enable, 57
Encoder
  Inkrementalencoder, 26
Endschalter, 55, 126
Endstufe, 38
EPOS Studio
  Kommunikation Tab, 17
  Tools, 18
  Wizards, 18
  Workspace Tab, 16

F

Fault, 56
Feedback, 32
Feed-Forward, 34
Firmware Specification, 65
Function, 98
Function Block, 97
  Enable, 97
  Execute, 97
Funktionsbausteine
  Zurücksetzen, 105

G

Genauigkeit
  Drehzahl, 50
Geschwindigkeit
  max. profile velocity, 46
Getriebe
  Elektronisches Getriebe, 59

H

Homing, 44
  Current threshold, 44
  With index channel, 45

I

I/O
  I/O Monitor, 53
IEC 61131-3, 74
IEC Programmiersprache
  Ablaufsprache AS (SFC), 80
  Funktionsbausteinsprache FBD, 84
  Strukturierter Text ST, 80
IEC programming language
  Function Block Diagram FBD, 116
Indexkanal, 26
Infoteam, 74
Installation
  EPOS Studio, 8
Instanz, 97
Interpolated Position Mode, 95

K

Kommunikation, 11
Kommutierung, 24, 126
  Block, 24
  Sinus, 24

L

LED
  Blaue LED, 75, 126
  grüne LED, 37
  rote LED, 37

M

Mask, 54
Maske, 53
  Ausführungsmaske (ExecMask), 54
Master, 11, 35, 126
Master Encoder Mode, 59, 95
MC_ Bibliothek, 95
MC_Direction, 87
MC_library
  MC_Home, 95, 112
  MC_MoveAbsolute, 95
  MC_MoveRelative, 95
  MC_MoveVelocity, 95, 108
  MC_Power, 95
  MC_Reset, 95
  MC_Stop, 95, 108
MCnegative, 87, 108
MCpositive, 87, 108
Monitor
  I/O Monitor, 53
Motor
  bürstenbehaftet (DC), 24
  bürstenlos (BLDC, EC), 24
MU_Bibliothek, 95
MU_library
  MU_GetDigitalInput, 115
  MU_SetDigitalOutput, 113
  MU_SetHomingParameter, 112

N

Network Configuration, 90

O

Objektfilter, 69
Objektverzeichnis, 64
  EPOS2 [internal], 67
  EPOS2 P, 65, 70
  Index, 64
Online kommandiert, 73
On-line kommandiert, 35
OpenPCS, 74
OSI-Modell, 62

P

Parameter
  Speichern, 69
Pfadgenerator, 39
PLC, 70
PLCopen, 74
Polaritöt der E/As, 53
Position Compare, 56

Position control
  Position Mode, 46
Position Marker, 55
Positionsregelung
  Profile Position Mode, 36
Program, 96
  cyclic task, 96
  interrupt task, 96
  Task, 96
  timer task, 96
Program Organization Unit, 96, 127
Programming Reference, 88, 94
Prozess-Daten-Objekt, 71
Prozess-Daten-Objekt
  Mapping, 71

Q

Quad counts, 26
Quick Stop, 55

R

Ready, 56, 126
Referenzfahrt, 44
Referenzschalter, 55
Resource
  Active resource, 92

S

Sample Project
  Simple Motion Sequence, 75
Schleppfehler, 34
  Max Following Error, 43
Schrittmotor, 59
Service-Daten-Objekt, 72
Setpoint Offset, 58
Setpoint Scaling, 58
Slave, 11, 35
Speed
  Maximum permissible speed, 22
SPS, 11
State machine
  SPS Programm als, 79, 80
Step Direction Mode, 59
Strom
  Nennstrom, 22
Stromregelung
  Current Mode, 52
Structured Text (ST)
  Syntax, 93
Strukturierter Text (ST), 80
Syntax check, 93
Syntax errors, 93

T

Task. vgl. Program
Time constant
  Thermal time constant, 22
Tuning, 28
  Auto Tuning, 28
  Expert Tuning, 29, 126
  Manuelles Tuning, 29

U

USB-Treiber, 10

V

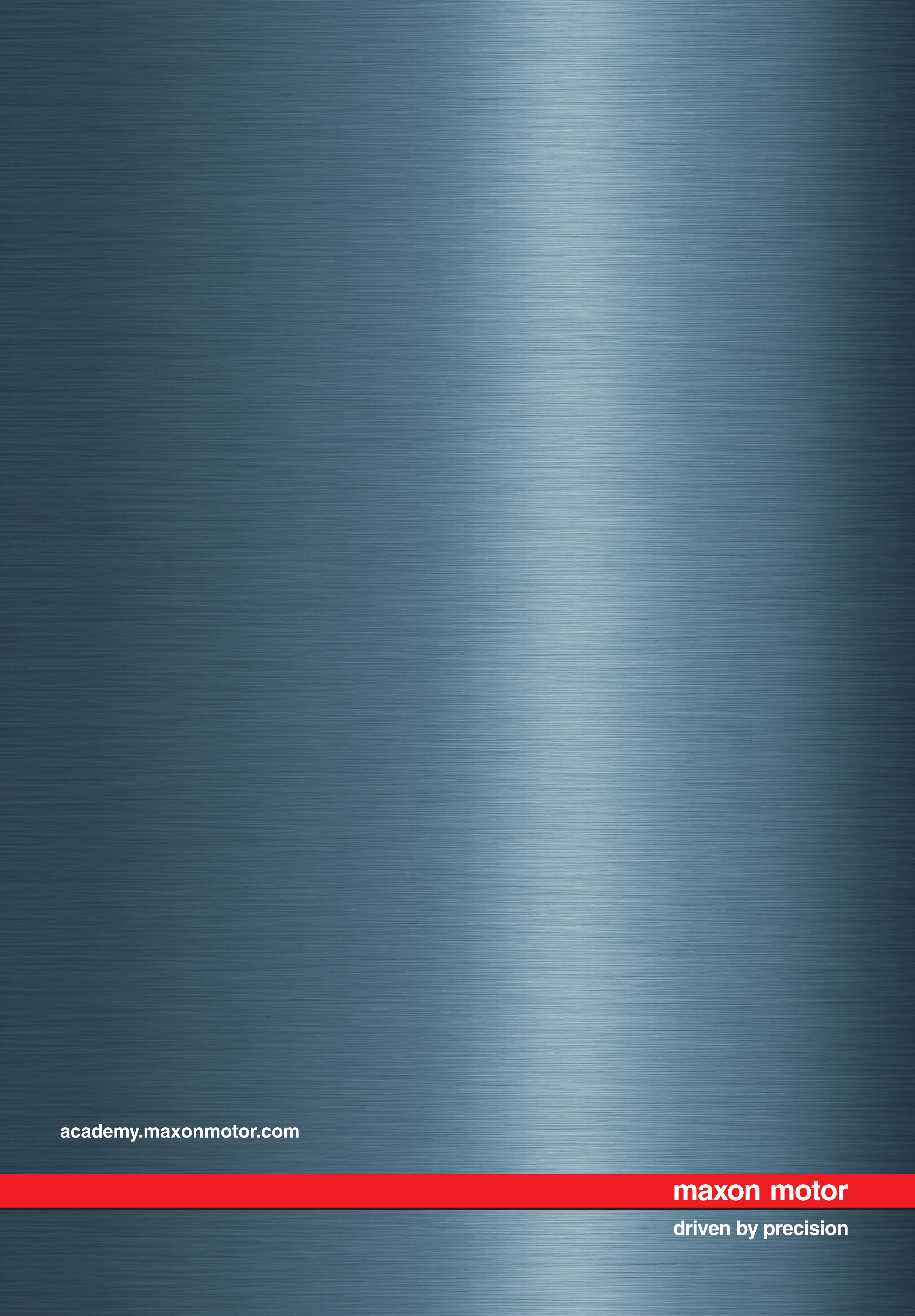
Variable, 86
  EPOS Datentypen, 87
  Global variables, 92
Verkabelung, 10
Vorsteuerung, 34

W

Wizard
  Installation Wizard, 8
  New Project Wizard, 14
  Parameter Export/Import wizard, 70
  Regulation Tuning Wizard, 28
  Startup Wizard, 19

Z

Zeitkonstante
  Thermische Zeitkonstante, 22
Zustandsmaschine
  CANopen, 63



academy.maxonmotor.com
maxon motor
driven by precision